Perttu Katainen

Mobilog-monialustaratkaisun laadunvarmistus

Metropolia Ammattikorkeakoulu

Insinööri (AMK)

Ohjelmistotekniikka

Insinöörityö

20.05.2014

Metropolia

Tekijä(t)	Perttu Katainen
Otsikko	Mobilog-monialustanratkaisun laadunvarmistus
Sivumäärä	34 sivua + 1 liite
Aika	20. toukokuuta 2014
Tutkinto	Insinööri AMK
Koulutusohjelma	Tietotekniikka
Suuntautumisvaihtoehto	Ohjelmistotekniikka
Ohjaajat	Lehtori Simo Silander, Tuotepäällikkö Samuli Nikkanen

Tässä insinöörityössä käsitellään monialustaisen mobiilisovelluksen asettamia haasteita sovelluksen tekniselle laadulle ja toteutukselle. Työ tehdään CGI Suomi Oy:lle käyttäen heidän kehittämää Mobilog-työnohjaussovellusta pohjaratkaisuna.

Mobilog on HTML5-pohjainen mobiilisovellus, jonka tehtävänä on tarjota työssään liikkuville työntekijöille joustava työnohjaussovellusratkaisu. Tämän insinöörityön keskeisenä teemana on löytää toimivia laadunvarmistuksen testausmenetelmiä ja ratkaisumalleja, joiden pohjalta tulevaisuudessa tuotettavan Mobilog-monialustasovelluksen laatu voidaan varmistaa. Työssä käydään läpi kattavasti erilaisten sovellusratkaisujen vahvuudet ja heikkoudet sekä perustellaan tehdyt valinnat sovelluksen toteuttamiseksi valituilla teknologioilla.

Esiteltyjen tekniikoiden pohjalta työssä luodaan sovelluskäännös jo olemassa olevasta Mobilog-sovelluksesta käyttäen Windows Phone 8 -mobiilikäyttöjärjestelmäalustaa. Käännöstyö toteutetaan siten, että se täyttää mahdollisimman hyvin työssä aiemmin esitellyt laatu ja testaus -kriteerit. Tarkoituksena on löytää keino HTML5-sovelluksen siirtämiseen joustavasti alustaratkaisulta toiselle, sekä ratkaisu NFC–toiminnallisuuden toteuttamiseen.

Kokonaisuutena työ tarjoaa kattavan lähtökohdan Mobilog-sovelluksen monialustaratkaisun toteuttamiselle tinkimättä sovelluksen laadusta. Varsinainen WP8-sovelluksen käännöstyö jäi vielä kesken ja sitä tullaan jatkamaan koko Mobilog-kehitystiimin voimin.

Avainsanat	HTML5, laadunvarmistus, testaus, Windows Phone 8, NFC, monialustaratkaisu, Mobilog

Authors	Perttu Katainen
Title	Quality Assurance of Mobilog Cross-platform Application
Number of Pages	34 pages + 1 appendices
Date	20. toukokuuta 2014
Degree	Bachelor of Engineering
Degree Programme	Information and Communications Engineering
Specialisation option	Software Engineering
Instructors	Simo Silander, Senior Lecturer, Samuli Nikkanen Product Manager

This Bachelor's thesis provides solutions for developing cross-platform mobile applications. The study was made for CGI Finland ltd using Mobilog Workforce management (WFM) application as the software platform. Mobilog is an HTML5-based mobile application that provides a flexible WFM-application for employees doing mobile work.

The main aim of the study was to find effective technologies to improve developing and testing in cross-platform mobile applications that meet the software Quality Assurance standards.

On the basis of the selected technologies the study describes how to create a cross-platform application versioning for Windows Phone 8. The versioning job focuses on a flexible transferring of the HTML5 code on to different operating systems and the implementation of the NFC reading feature on Windows Phone 8. The application versioning had to fulfill all the quality and testing standards which are presented in the thesis.

The study provides comprehensive guidelines for developing cross-platform applications. Windows Phone 8 application development will continue in the future and it will be done by the Mobilog development team.

Keywords	HTML5, WFM, Cross-platform, Quality Assurance, NFC, Windows Phone 8, Mobilog, Testing

Sisältö

Sanasto

Android .apk Googlen suunnittelema paketointi Android-sovelluksia varten.

Apache-lisenssi Apache Software Foundationin luoma vapaan ohjelmiston lisenssi.

BSD-lisenssi Vapaa ohjelmistolisenssi(Berkey Software Distribution).

CoffeeScript JavaScript-muunnos.

Cordova/PhoneGap Soveluskehys HTML5-sovelluksen kehitystä varten.

Extreme Programming Menetelmä jossa kehittäjät kehittävät pareittain sovellusta.

GIT Ilmainen ohjelmakoodin versionhallintasovellus.

GNU/GPL GPL Avoimen lähdekoodin lisenssi (General Public Licence).

HTML5 Web-pohjainen sovellusteknologia.

Hyväksymistestaus Sovelluksen testauksen viimeinen vaihe ennen sen julkaisua.

iOS Applen kehittämä mobiili käyttöjärjestelmä.

JavaScript HTML5-sovelluksissa yleisesti käytetty ohjelmointikieli.

Jenkins Jatkuvan testauksen työkalu.

JQuery JavaScriptiin pohjautuva käyttöliittymäkirjasto.

Käyttötesti Testitapaus, jossa sovellusta käytetään sen todellisessa ympäristössä.

MIDlet Pelkistetyllä Javalla luotu sovellustekniikka.

Mobilog CGI Suomi Oy:n kehittämä mobiili -työnohjausjärjestelmä.

NFC Tiedonsiirtotekniikka käytettäessä lyhyitä lukuetäisyyksiä.

Monialustasovellus Sovellustyyppi, joka toimii usealla eri päätelaitteella.

Scrum Toimintamenetelmä ketterää sovelluskehitystä varten.

TDD Testivetoinen kehitysmalli sovellusten tuottamiseen.

Toiminnallisuuden testaus Toiminnallisuuden testaaminen omana yksikkönään.

Virheindeksi Mahdollistaa tuotteen virheiden tilastoimisen sekä niiden analysoimisen.

Visual Studio Microsoftin kehittämä editori koodin kirjoittamiseen.

WFM Työnohjausjärjestelmä (Workforce Management)

Windows Phone 8 / WP8 Microsoftin suunnittelema mobiili käyttöjärjestelmä.

1 Johdanto

Tämä insinöörityö tehdään CGI Suomi Oy:lle kartoittamaan mobiilin monialustaratkaisun laadunvarmistukseen liittyviä haasteita. CGI on Kanadalaislähtöinen ohjelmistoalalla toimiva konsultointiyritys, jolla on useita toimipisteitä ympäri Suomea. Pääsääntöisesti CGI palvelee suuria asiakkuuksia, kuten kaupunkeja ja kuntia. Työn pohjana käytetään CGI Suomi Oy:n tuottamaa Mobilog-työnohjausjärjestelmän HTML5-pohjaista mobiilisovellusta.

Tämän työn tavoitteena on tuottaa kattava selvitys, mitä haasteita Mobilog HTML5 -sovelluksen kääntäminen usealle käyttöjärjestelmäalustalle asettaa ja miten useaan käyttöjärjestelmäalustaan siirtyminen vaikuttaa tuotteen testaamiseen. Työssä käydään läpi menetelmiä, jotka pyrkivät parantamaan sovelluksen laatua monialustaisessa toteutuksessa. Pääpaino tässä insinöörityössä on erilaisten laadunvarmistustekniikoiden sekä sovellusteknologioiden käyttömahdollisuuksien selvittäminen monialustaisissa mobiiliratkaisuissa. Tämän insinöörityön tavoitteena on löytää tasapaino sovelluksen laadun ja alustalevinneisyyden välillä.

Työssä esiteltyjen tekniikoiden avulla toteutetaan jo olemassa olevasta Mobilog-Android-sovelluksesta Windows Phone 8 -sovelluskäännös. Sovelluskäännöksen tarkoituksena on selvittää Windows Phone 8 -sovelluspaketointiratkaisun toimivuus Mobilog–sovelluksen pohjaratkaisuna, ottamatta kantaa jo valmiiksi toteutettuun HTML5 -koodiin. Työssä esiteltyjen menetelmien ja arkkitehtuuriratkaisuiden soveltamista sovelluksen tuottamiseen käsitellään ainoastaan sovelluspaketoinnin kohdalla. Jatkoa ajatellen työn on tarkoitus ohjata kehitystyötä ja muuttaa ohjelmointikäytäntöjä koko kehitystiimin osalta. Lopputuloksena tämä insinöörityö mahdollistaa aiheen jatkokehityksen sekä tarjoaa lähtökohdan laadun parantamiseksi Mobilog-sovelluksessa.

2 Mobilog-työnohjausjärjestelmä

2.1 Mobilog lyhyesti

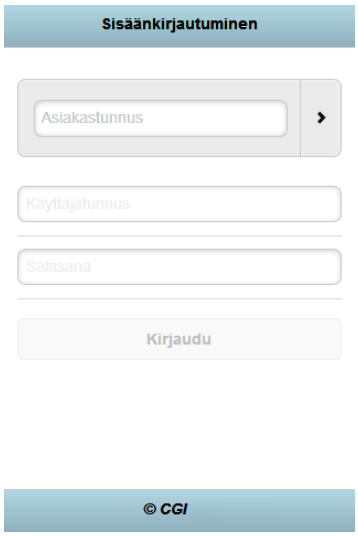
Mobilog on CGI Suomi Oy:n kehittämä toiminnanohjausjärjestelmä, jonka tehtävänä on palvella työssään paljon liikkuvia työntekijöitä kuten vartijoita, kodinhoitajia ja siivoojia. Järjestelmän avulla voidaan suorittaa työajan seurantaa, tuntikirjausta sekä tuottaa raportteja asiakkaiden tarpeisiin. Toiminnanohjausjärjestelmää voidaan käyttää tarvittaessa itsenäisenä toiminnanohjaustuotteena tai se voidaan vaihtoehtoisesti integroida yhteen erilaisten taustajärjestelmien kanssa. Mobilog on mobiilisovellus, joka hyödyntää muun muassa puhelimen kameraa sekä *NFC*-toiminnallisuutta työmääräysten tiedon keräämiseen suoraan työkohteesta. Mobilogin menettelytapa mahdollistaa tehokkaamman toimintamallin työaikatietojen keräämiseen verrattaessa perinteiseen työnohjausjärjestelmään, jossa työntekijät kirjaavat työsuoritteitaan työvuoron jälkeen toimistolla. Mobilogin päätarkoituksena on luoda joustava työnohjaus ja toimintaympäristö niihin työtehtäviin, joissa työtehtävät suoritetaan päivän aikana useissa kohteissa.

2.2 Mobilog-työnohjausjärjestelmän rakenne

Mobilog-työpöytäsovellusta käytetään verkkoselaimella, jolloin erillistä sovellusasennusta tietokoneelle ei tarvita. Sovelluskokonaisuus tarjotaan asiakkaalle CGI:n hallinnoimilta palvelimilta, joita Mobilog-tiimi ylläpitää Helsingistä käsin. Mobilog -mobiilisovellus toimii tällä hetkellä Android-käyttöjärjestelmän omaavissa päätelaitteissa, ja se täytyy asentaa erikseen työntekijän puhelimeen. Mobilogista on olemassa myös vanhempiin Nokia-malleihin suunniteltu MIDlet-sovellus, joka sisältää pääpiirteittäin samat toiminnallisuudet kuin Android-sovellus. Mobilog-sovelluskokonaisuus voidaan jakaa kolmeen pääosa-alueeseen jotka ovat Työnohjaussovellus, mobiilisovellus sekä liitettävät taustajärjestelmät eli sovellusintegraatiot.

2.2.1 Mobilog-mobiilisovellus

Mobilog-mobiilisovellus on toteutettu käyttämällä HTML5-pohjaisia teknologioita sekä Android .apk-ohjelmapaketointia. Mobiilisovellus on kehitetty hyödyntämään päätelaitteiden tarjoamia laitteita työsuoritteiden tiedon keräämiseen, kuten *GPS* (paikannus), *NFC* (tunnisteen luku) ja kamera (*QR* -tunnisteen luku). Käyttäjä voi tunnisteita lukemalla aloittaa ja lopettaa työaikansa sekä kuitata työmääräyksen työtehtäviä tehdyiksi. Riippuen tunnisteen tyypistä voi käyttäjä lukea sen vaihtoehtoisesti joko puhelimen kameralla tai hyödyntäen NFC-luentaa. Mobilog-sovelluksesta löytyy myös mahdollisuus kirjata työaikasuoritteita käyttäen pelkkää mobiilisovellusta, mikäli puhelin ei tue esimerkiksi NFC-ominaisuutta. Kun työmääräys on kuitattu valmiiksi, lähettää sovellus sen automaattisesti Mobilog-taustajärjelmään. Sovelluksen käyttöliittymä on suunniteltu palvelemaan eri ammattikuntia, ja sen termistö on yleispätevää, joten erillisiä asiakaskohtaisia näkymiä käyttöliitymästä ei tarvita.

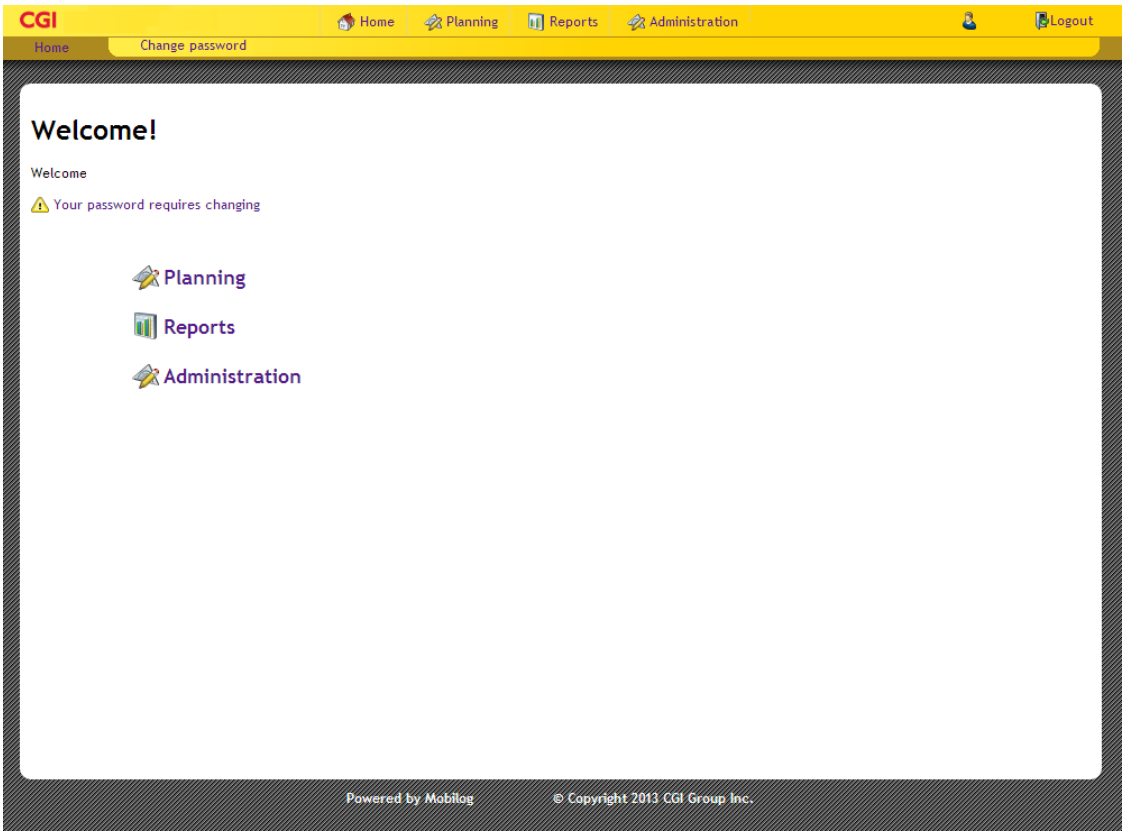


Kuva 1: Mobiili käyttöliittymä

Kuvassa 1 esiintyvä mobiilisovelluksen käyttöliittymä on toteutettu käyttäen *JQuery* -kirjastoa, ja sen toiminnallinen logiikka on kirjoitettu *CoffeeScriptillä*. NFC ja kameratoiminnallisuudet vaativat erikseen Javalla kirjoitetun sovellusliitännäisen sekä Androidin tarjoaman .apk-sovelluspaketoinnin.

2.2.2 Mobilog-työnohjaussovellus

Mobilog-työnohjaussovelluksen toiminnallinen logiikka on toteutettu pääsääntöisesti käyttäen Java-ohjelmointikieltä. Käyttöliittymän toteutuksessa on hyödynnetty *JavaScript*-, sekä *JQuery*-tekniikoita. Sovellusta käytetään verkkoselaimella ja sen päätarkoituksena on palvella esimiesten sekä työnsuunnittelijoiden tarpeita. Työnohjaussovellus mahdollistaa työmääräyksen allokoinnin ja lähetyksen työntekijöille, sekä sen avulla voidaan seurata töiden edistymistä reaaliajassa (kuva 2). Käyttöliittymän toiminnallisuuksia sekä esillä olevien ikkunoiden näkyvyyttä voidaan konfiguroida asiakkaiden toiveiden mukaisesti. Tämä mahdollistaa työnohjaussovelluksen tehokkaan muuntautumiskyvyn asiakkaan tarpeisiin.

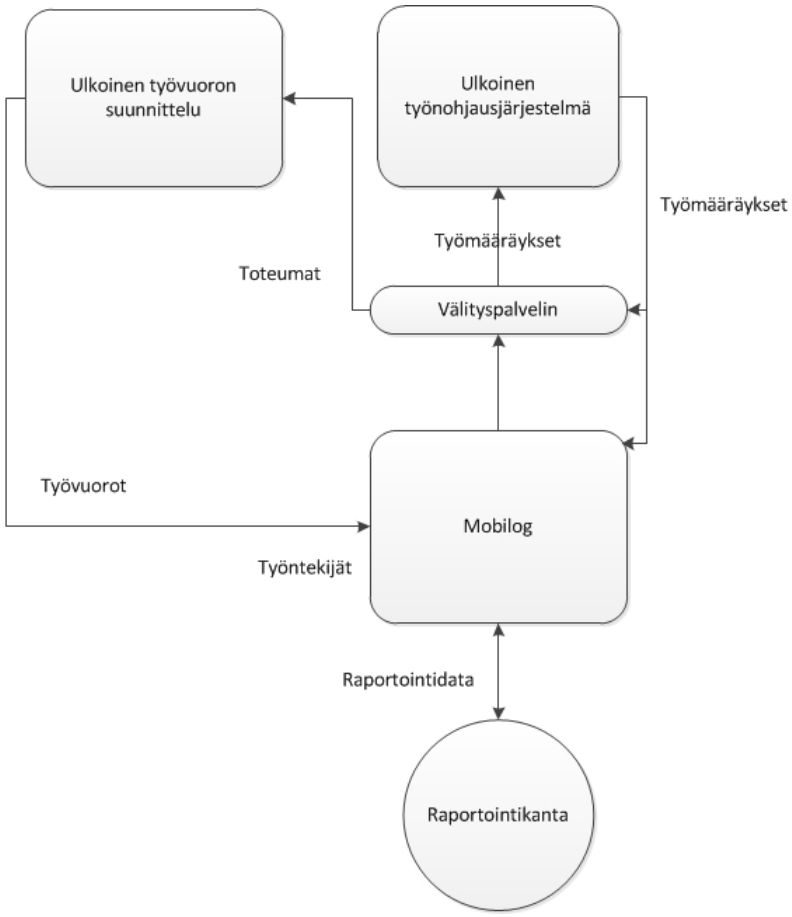


Kuva 2: Työnohjausjärjestelmä.

Työnohjaussovelluksen käyttäjäoikeudet on jaettu kahteen osa-alueeseen: työnsuunnittelijoihin sekä pääkäyttäjiin. Työnsuunnittelijat hoitavat ainoastaan työmääräysten käsittelyjä ja valvovat lähetettyjen työmääräysten toteutumista. Pääkäyttäjät voivat tuottaa raportteja, hallinnoida työntekijöitä, asiakkaita sekä tehdä työnsuunnittelijoiden työtehtäviä. Pääkäyttäjille sovellus tarjoaa työntekijöiden sekä integraatioiden hallintaan tarkoitetun näkymän, josta koko ohjelman hallinnointi tapahtuu.

2.2.3 Mobilogiin liitettävät integraatiot

Mobilog voidaan kytkeä integraatioiden avulla erilaisiin taustajärjestelmiin, jolloin se tarjoaa mobiilitoiminnallisuuden jo asiakkaalla käytössä oleviin työnohjausjärjestelmiin. Integraatioiden avulla esimerkiksi työvuorosuunnittelut ja työmääräykset (kuva 3) voidaan toteuttaa toisessa työnohjausjärjestelmässä, jolloin Moblilog välittää sille lähetetyn vuorotiedon automaattisesti työntekijän puhelimeen. Haluttaessa Mobilogista voidaan palauttaa integraatioiden kautta laskutustietoa, raportointikantoja tai tehtyjä toteumia takaisin työmääräyksen lähettäneeseen työnohjaussovellukseen. Integraatioiksi voidaan laskea erilaisia taustajärjestelmiä kuten työnohjaussovelluksia, ulkoisia työvuoron suunnittelusovelluksia tai raportointijärjestelmiä (kuva 3). Integraatioita voi olla yhdessä Mobilog-järjestelmässä useita, mikä mahdollistaa sen jouhevan toiminnan osana suurempaa sovelluskokonaisuutta. Yleisesti integraatiot liitetään suoraan Mobilogiin rajapintatoteutusten kautta, mutta välityspalvelinten käyttö välikappaleena on myös yleinen ratkaisu tiedon välittämiseen.



Kuva 3: Integraatiomalli.

3 Ongelman esittely

Mobilog-mobiilisovelluksen lähtökohtana on pystyä tarjoamaan asiakkaille sovellus, joka toimii kaikilla asiakkaan valitsemilla päätelaitteilla. Monialustainen sovellustoteutus asettaa laadulle useita teknisiä haasteita, sillä monialustaisen sovelluksen on tarjottava yhdenmukainen toiminnallisuus riippumatta puhelimen mallista tai käyttöjärjestelmästä. Laadun kannalta tarkasteltuna tämä tarkoittaa myös sitä, että sovelluksen käyttökokemuksen täytyy olla yhdenmukainen. Haasteena usean sovellusalustan ylläpidossa sekä kehityksessä on erityisesti uusien puhelinmallien sekä käyttöjärjestelmien nopea kehitys. Mobiilisovellukset ovat tuoretta teknologiaa IT-alalla, joten myös standardit elävät vielä huomattavasti nopeammin kuin esimerkiksi perinteikkäässä PC-maailmassa. Työn tarkoituksena on löytää mahdollisimman hyvä ja vakaa menetelmä luoda toimiva monialustaratkaisu Mobilog-sovellukseen sekä samalla säilyttää koko tuotteen laatu.

3.1 Monialusta-mobiilisovellus

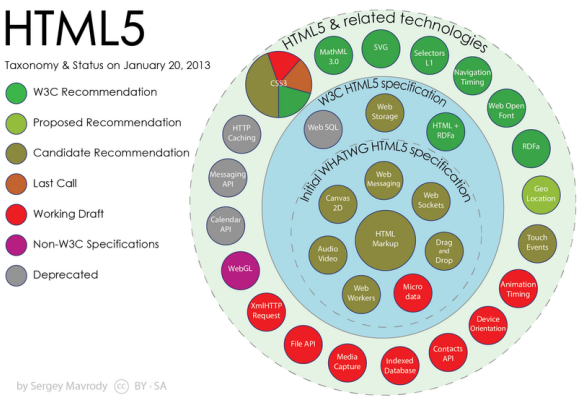
Monialustainen mobiilisovellus on sovellustyyppi, joka toimii nimensä mukaisesti usealla käyttöjärjestelmäalustalla. Pääsääntöisesti monialustaiset sovellukset voidaan jakaa kahteen luokkaan: *aidosti monialustaisiin sovelluksiin* sekä *jokaiselle ympäristölle erikseen käännettäviin sovelluksiin.* [1] Aidosti monialustainen sovellus toimii alustaratkaisusta riippumatta millä tahansa päätelaitteella. *HTML5*-teknologia mahdollistaa aidosti monialustaisen sovelluksen luomisen, sillä sen tekninen toteutus ei ota kantaa alla olevaan käyttöjärjestelmään. Jokaiselle ympäristölle erikseen käännettävä sovellus (*Natiivi sovellus*) joudutaan kehittämään käyttäen valitun käyttöjärjestelmän tukemaa ohjelmointikieltä sekä ohjelmistokirjastoja. Mikäli sovellus halutaan saada toimimaan muilla käyttöjärjestelmäalustoilla, joudutaan yleisimmin koko sovelluslogiikka sekä käyttöliittymäkoodi kirjoittamaan kokonaan uudestaan. Eräänlaisena hybridiratkaisuna on käyttää molempien tekniikoiden ominaisuksia, mikäli käytettävä tekniikka ei yksinään kykene tarjoamaan riittävää toiminnallisuutta. Yleisimmin mobiilit HTML5-sovellukset ovat juuri hybridi-ratkaisuja, tekniikan tarjoaman joustavan kehityksen myötä.

3.2 Erilaiset sovellusarkkitehtuurit mobiilikehityksessä

Tällä hetkellä mobiilisovelluksille on tarjolla useita käyttöjärjestelmävaihtoehtoja. Yleisimmin nykyiset mobiilipohjaiset käyttöjärjestelmät tukevat täysimittaisesti *HMTL5*-sovellustoteutuksia, mutta niiden käyttämät ohjelmapaketointiratkaisut sekä ohjelmointikielet poikkeavat toisistaan.

3.2.1 HTML5-sovellus

HMTL5 on *HTML*-kielen tuorein versio, jolla voidaan toteuttaa verkkosivuja sekä sovelluksia. Yleisellä tasolla termi *HTML5* viittaa uusiin Web-teknologioihin (kuva 4), joita ovat muun muassa *CSS3*, *JavaScript*, *JQuery* ja *Geolocation API*. Tuki erilaisille teknologioille on laaja, sillä *HTML5*-standardia ei ole vielä virallistettu. Siitä on olemassa ainoastaan W3C:n suositusehdokas, jonka pohjalta kielen standardi on tarkoitus vahvistaa vuoden 2014 lopussa.[2]



Kuva 4: HTML5-tuetut teknologiat.

Yleisesti *HTML5*-sovellus toteutetaan käyttäen *JavaScript*-, *CSS3*-tekniikoita sekä erilaisia sovelluskirjastokokoelmia. Itse *HTML*-kielen osuus koodissa jää monesti varsin pieneksi, joten kielikuva pelkästä *HTML5*-sovelluksesta on hieman harhaanjohtava. HTML5-sovellus voidaan toteuttaa joko stand alone (omana sovelluksenaan) tai verkkoon verkkosivupohjaisena Web-sovelluksena, mikä mahdollistaa sen monipuolisen käytön erilaisissa ratkaisuissa. HTML5-tekniikan vahvuutena on sen tekninen tuoreus, ajonaikainen käännettävyys sekä erittäin vahva alustariippumattomuus. Sen heikkoutena voidaan pitää erityisesti suoraa komponenttitukea, joka joudutaan tekemään esimerkiksi PhoneGapin avulla. Tarkempi tuettujen komponenttien talulukko esitetään luvun 3 kuvassa 6.

3.2.2 Android .apk -sovellus

Android .apk -sovelluspaketointi on Googlen Android-puhelimissa käytettävä formaatti. Teknisesti ratkaisu on hyvin samankaltainen kuin esimerkiksi *MSI* tai Debian Linuxin *.deb -paketointi*. Android apk –pohjainen sovellus muodostetaan ensin kääntämällä lähdekoodit tavukoodiksi ja sen jälkeen paketoimalla käännetty data yhdeksi tiedostoksi [3]. Tuettuja ohjelmointikieliä ovat muun muassa *Java*, *C++* ja *XML*. HTML5-sovellus voidaan liittää suoraan Android .apk -sovellukseen käyttäen *PhoneGap* -sovelluskehystä, tai vaihtoehtoisesti luomalla pohjaratkaisu käyttäen *XML/XHMTL5* -tekniikoita [4]. Android -kehitystyökalut tarjoavat ohjelmoijalle valmiin API:n, jota vasten kehittämällä voidaan varmistua, että sovellus toimii teknisesti oikein Android apk –pohjaisissa laitteissa. Android on maailman laajalevikkisin mobiilialusta (31.3.2014), joten sovelluksen kehittäminen .apk-formaattiin on kannattavaa. Toisaalta alusta on jakautunut jo useampaan versiohaaraan, joiden pääsuunnat ovat *2.x.x* ja *4.x.x*, mikä mutkistaa hieman sovelluskehitystä, sillä kaikki ominaisuudet eivät toimi ristiin eri alustaversioiden välillä. Osittain Android-sovelluksen vahvuutena on sen ilmainen kehitysympäristö sekä levitettävyys. Androidiin pohjautuvaa laitetta ei tarvitse asettaa sovellusta kehitettäessä erikseen *kehittäjätilaan* ja sovellusta voi jakaa vapaasti ilman virallisia levityskanavia. Kehitettyä sovellusta on myös mahdollista halutessaan levittää Google Play -sovelluskaupan kautta.

3.2.3 Windows Phone 8 -sovellus

Windows Phone 8 (*WP8*) on Microsoftin tuottama mobiililaitteille suunnattu käyttöjärjestelmä. Se pohjautuu Microsoftin standardeihin sekä työkaluihin ja sille kehitetty sovellus on yhteensopiva yhtiön muiden tuotteiden kanssa. Järjestelmä on suljettu, joten sovelluksen ohjelmointikieliksi sekä kehitysympäristöiksi kelpaavat pääsääntöisesti Microsoftin tarjoamat teknologiat. Kokonaisuutena WP8-alusta on osa Microsoftin sovellusekosysteemiä, joka muistuttaa paljolti Applen vastaavaa mallia. Sovelluskehityksessä käytettäviä ohjelmointikieliä ovat *C#* sekä *Visual Basic*, mutta WP8-alusta tukee myös Androidin tapaan *HTML5*-standardien mukaisia sovelluksia, sillä osa avoimen lähdekoodin lisensseistä ovat sallittuja sovelluskehityksessä kuten *Apache* ja *BSD* [5]. Poiketen Androidin tavasta levittää vapaasti sovelluksia WP8 -sovelluksia voidaan levittää ja asentaa ainoastaan Microsoft Phone Marketplace -sovelluskaupan avulla. Sovellusten vapaa kehitys suoraan

laitteelle on estetty, joten kehitysvaiheessa olevaa sovellusta voidaan testata ainoastaan *Visual Studio IDE*:n emulaattorilla, tai hankkia vaihtoehtoisesti Microsoftin tarjoama kehittäjälisenssi, jolloin sovellusta on mahdollista testata suoraan puhelimella.

3.2.4 iOS-sovellus

iOS on Applen kehittämä käyttöjärjestelmä yhtiön omiin mobiililaitteisiin. Järjestelmä perustuu *Darwin BSD* –käyttöjärjestelmään sekä yhtiön siihen tekemiin omiin komponentteihin [6]. Sovelluskehitys tehdään pääasiallisesti käyttäen Applen omaa *Objective-C*-ohjelmointikieltä sekä muita tuettuja teknologioita, kuten *PhoneGap:ia* ja *BSD:tä*. Androidin ja Windows Phonen tapaan sille voidaan myös kehittää *HTML5*-sovelluksia käyttäen pohjaratkaisuna Applen omaa sovelluspaketointia. Itse käyttöjärjestelmä on Androidin ja Windows Phonen välimaastossa avoimuuden suhteen. BSD on oma OpenSource-lisenssinsä [7], jonka periaatteena on tarjota yksinkertaisempi ja vapaampi ohjelmistolisenssi kuin esimerkiksi *GNU GPL* (GNU General Public Licence). Vaikka vain osa käyttöjärjestelmän koodista on Applen sulkemaa, sovelluksien julkaisu tätyy tehdä käyttäen Applen tarjoamaa virallista *AppStore*-julkaisukanavaa.

3.2.5 Muut mobiilialustat

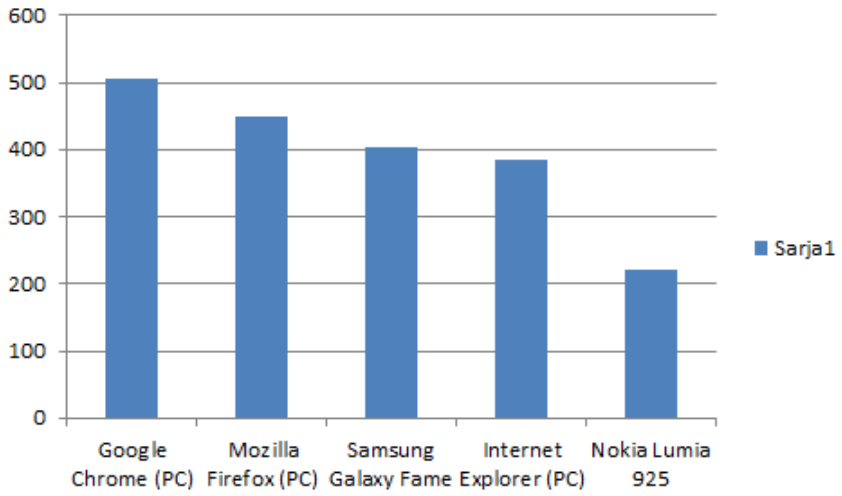
Markkinoille ilmestyy tasaisin väliajoin valtavirrasta poikkeavia järjestelmiä, jotka yleisimmin pohjautuvat löyhästi Linuxiin. Pienet alustat tarjoavat varsin hyvät lähtökohdat kehittää sovelluksia haluamillaan teknologioilla, sillä standardit kyseisissä järjestelmissä ovat väljemmät. Uusimmat tulokkaat *Jolla Sailfish* sekä *Firefox OS* ovat molemmat ilmaisia sekä avoimia mobiilikäyttöjärjestelmäratkaisuja, joihin ei tarvitse erikseen kehittäjälisenssejä. Firefox OS luottaa sovelluksissaan puhtaasti *HTML5*-teknologiaan, tarjoten samalla kommunikointimahdollisuuden suoraan puhelimen laitteiston kanssa [8]. Suomalainen Jolla tarjoaa vapaamman sovelluskehitystavan tarjoten tuen *QT*-, *C++*- ja jopa Android-teknologioille sovelluksen kehittämiseksi [9]. Markkinoilla on tarjolla lisäksi lukuisia muita mobiilikäyttöjärjestelmiä, mutta niihin ei oteta kantaa tässä työssä. Uudet valtavirrasta poikkeavat käyttöjärjestelmäalustat tarjoavat useassa tapauksessa paljon uusinta teknologiaa, mutta varsin vähän tukea, joten riski kehittää sovellusta pelkästään näille järjestelmille on suuri.

3.3 HTML5 ja natiivin sovelluksen eroavaisuudet

Pelkillä *HTML5*-tekniikoilla toteutettu sovellus on aidosti monialustainen sovellusratkaisu, sillä käyttöliittymän sekä sovelluslogiikan toteutus ei riipu alla olevasta käyttöjärjestelmästä. Tämän vuoksi puhtaasti *HTML5*-pohjaisen sovelluksen siirtäminen alustaratkaisulta toiselle on helppoa. HTML5-tekniikoilla toteutettu sovellus mahdollistaa käytännössä sovelluskehittäjien osallistumisen saman koodipohjan kehittämiseen, mikä tekee ohjelmointityöstä suoraviivaista ja kehittäjien siiloutumiselta vältytään. Siiloutuminen tarkoittaa kehittäjäjoukkojen jakautumista ainoastaan tiettyjen teknologioiden tai alustaratkaisujen pariin, mikä taas voi johtaa sovelluskehityksen hajautumiseen kehitystiimissä. Verrattaessa *HTML5*-sovelluksen kehitystyötä *natiivin sovelluksen* kehitystyöhön, tulee vastaan selkeä ero. Natiivi sovellus joudutaan kääntämään jokaiselle alustaratkaisulle erikseen. [1] Sovelluksen kääntäminen voi helposti muodostua varsin työlääksi, elleivät erilaiset käyttöjärjestelmät tue samaa ohjelmointikieltä. Verratessa esimerkiksi Windows Phonen ja Androidin tukemia ohjelmistotekniikoita huomataan ,että koodin uudelleenkierrättäminen alustalta toiselle on minimaalista, sillä käyttöjärjestelmien arkkitehtuurit, rajapinnat ja ohjelmistokirjastot poikkeavat paljon toisistaan.

Selkeänä haittapuolena *HTML5*-teknologiassa voidaan havaita sen vaihteleva tuettavuus käytettävissä mobiililaitteissa. Vielä keskeneräinen *HTML5* -standardi on tuettu lähes kaikissa moderneissa mobiililaitteissa, mutta ainoastaan osittain. Suoritin kokeita erilaisilla verkkoselaimilla ja sain yllätykseksenі selville, että tuettujen *HTML5*-ominaisuuksien määrä vaihtelee niissä varsin radikaalisti. Kuten kuvan 5 diagrammista käy ilmi, PC:llä tekemäni *HTML5*-testi Google Chrome verkkoselaimella antoi pisteiksi 505/555, kun taas WP8 -käyttöjärjestelmällä varustettu Nokia Lumia 925 sai vastaavassa testissä vain 222/555 pistettä. Testisivu erottelee eri osa-alueet tuettujen ominaisuuksien mukaan ja samalla pisteyttää ne. Testin voi käydä tekemässä osoitteessa http://html5test.com/. Testitulosten erojen suuruus kertoo osittain karua kieltä puhtaasti *HTML5* -teknologiaa käyttävistä sovelluksista. Käytännössä tämä tarkoittaa sitä, että täyttä varmuutta *HTML5*-sovelluksen toiminnalle ei voida vielä taata siirryttäessä alustalta toiselle.

Tarkastellessa asiaa natiivin sovelluksen näkökulmasta ei vastaavanlaista ongelmaa suoranaisesti esiinny. Natiivi sovellus kirjoitetaan käyttäen järjestelmän tarjoamia ohjelmistokomponentteja sekä rajapintoja, jolloin jokaiselle käyttöjärjestelmäalustalle luotu sovellus voidaan optimoida käyttöjärjestelmäkohtaisesti [10]. Optimoinnilla saavutetaan sovellukseen parempi suorituskyky kuin vastaavanlaisella *HTML5*-toteutuksella sekä taataan sovelluksen toimivuus kaikissa saman käyttöjärjestelmäversion omaavissa laitteissa. Natiivi mobiilisovellus myös mahdollistaa sitä käyttävän mobiililaitteen komponenttien suoran käytön sovelluksessa. Suora yhteys laitteen komponentteihin helpottaa varsinkin puhelinten *NFC*- tai *kamera*-liitännäisen toteuttamisessa.



Kuva 5: HTML5-tuen testaus.

Tällä hetkellä *HTML5*-standardi ei tue suoraa laitteistointegraatiota, vaan yhteys päätelaitteen komponentteihin muodostetaan esimerkiksi *PhoneGap:n* avulla. PhoneGap ei kuitenkaan tue täysmittaisesti nykyistä lisälaitevalikoimaa, vaan se pyrkii tarjoamaan mahdollisimman hyvän tuen mahdollisimman monelle laitteelle. Vaikka kuvan 6 taulukko osoittaa kameran olevan tuettu PhoneGap:n kautta, paljastui, että sen toiminta on varsin rajoittunutta. Kameran toiminta on mahdollista ainoastaan *snapshot*-periaatteella, eli ainoastaan valokuvan tuominen suoraan laitteelta HTML5-sovellukseen onnistuu. Liikkuvan kuvan siirtoa puhtaaseen *HTML5*-sovellukseen ei ole vielä toteutettu. Sama ongelma pätee *NFC*-piirien toimintaan, sillä suoraa tukea *NFC*-laitteille ei löydy vielä nykyisestä PhoneGap:n versiosta. Muilta osin PhoneGap tarjoaa varsin kattavan laitetuen useimmille käyttöjärjestelmäalustoille, ja se on helppo integroida uusiin ohjelmistoprojekteihin.

	iPhone / iPhone 3G	iPhone 3GS and newer	Android	Blackberry OS 6.0+	Blackberry 10	WebOS	Windows Phone 7 + 8	Symbian	Bada
Accelerometer	✓	✓	✓	✓	✓	✓	✓	✓	✓
Camera	✓	✓	✓	✓	✓	✓	✓	✓	✓
Compass	x	✓	✓	x	✓	✓	✓	x	✓
Contacts	✓	✓	✓	✓	✓	x	✓	✓	✓
File	✓	✓	✓	✓	✓	x	✓	x	x
Geolocation	✓	✓	✓	✓	✓	✓	✓	✓	✓
Media	✓	✓	✓	x	✓	x	✓	x	x
Network	✓	✓	✓	✓	✓	✓	✓	✓	✓
Notification (Alert)	✓	✓	✓	✓	✓	✓	✓	✓	✓
Notification (Sound)	✓	✓	✓	✓	✓	✓	✓	✓	✓
Notification (Vibration)	✓	✓	✓	✓	✓	✓	✓	✓	✓
Storage	✓	✓	✓	✓	✓	✓	✓	x	x

✓ - supported feature
x - unsupported feature due to hardware or software restrictions

Kuva 6: PhoneGap:n tukemat komponentit.

Molemmissa toteutustavoissa käyttöliittymän suunnittelu on varsin vapaata hyvän kirjastotarjonnan takia. HTML5-ratkaisu on tekniseltä toteutukseltaan perinteisempi ja avoimempi johtuen sen *HTML*-kielisistä juurista. Sovelluksen niin sanottu ”Look and feel” määräytyy siis täysin sovelluskehittäjän oman halun mukaan. CSS3 tarjoaa *HTML5* -sovellukseen tehokkaat visuaaliset työkalut sovelluksen ulkonäön määrittämiseen sekä tarvittaessa animointiin. Natiivissa mobiilisovelluksessa käytetään yleisesti järjestelmän valmistajan tarjoamia grafiikkakomponentteja käyttöliittymän toteuttamiseen. Toiminnallisuudet ovat valmiiksi testatut ja hiotut tarkoitukseen, mikä vähentää virheellisen koodin määrää käyttöliittymän toteutuksessa. Toisaalta käytettäessä valmiita grafiikkakirjastoja ei sovelluksen yleiseen ulkonäköön pysty juuri vaikuttamaan. Esimerkiksi Windows Phonen ja Androidin tarjoamat vakiokirjastot eroavat todella paljon toisistaan ja saman toiminnallisuuden omaavat sovelluset voivat näyttää täysin erilaisilta eri valmistajan laitteissa.

Hybridimobiilisovellus on kahden edellä vertaillun teknologian yhdistelmä. Pyrkimyksenä hybridisovelluksessa on tarjota *HTML5*-kielen joustavuus sekä natiivin sovelluksen järjestelmään integroitavuus [10]. Sovelluslogiikka ja käyttöliittymä voidaan toteuttaa käyttämällä *HTML5*-tekniikan tarjoamia toiminnallisuuksia ja puutteet sovelluksen komponenttituessa voidaan korvata natiivin sovelluksen keinoin. Mobilog-mobiilisovellus on toteutettu käyttäen hybridisovellusteknologiaa, sillä mobiililaitteen komponenttien tuki on keskeisessä roolissa sovelluksen toiminnassa.

3.4 Monialustaisen sovellusratkaisun asettamat haasteet laadulle

Monialustainen mobiilisovellus asettaa joka tapauksessa teknisen laadunvarmistuksen uuden haasteen eteen, valitaan toteutustekniikaksi mikä tahansa aiemmin esitellyistä vaihtoehdoista. Kun asiaa tarkastellaan sovelluksen ylläpidon näkökulmasta, olisi puhtaasti *HTML5*-pohjainen sovellusratkaisu helpommin hallittavissa ja kehitettävissä. Yhden koodipohjan testaaminen ja kehittäminen on huomattavasti suoraviivaisempaa, mikä tarjoaa testaajille ja kehittäjille enemmän aikaa keskittyä sovelluksen kehitystyöhön. Aidon monialustasovelluksen malli pätee tosin ainoastaan puhtaaseen *HTML5*-sovellukseen. On muistettava, että yleisesti *HTML5*-sovellusratkaisu käyttää useita ohjelmointikieliä sekä kolmannen osapuolten tekniikoita toiminnassaan, jotka lisäävät kattavan testauksen taakkaa. Sovelluksen laadun yhtenä kulmakivenä voidaan pitää sovellustestausta [11]. Sovellustestauksen tarkoituksena on paikantaa sovelluksessa ilmenevät viat, jotta ne voidaan korjata sovelluskehittäjien toimesta. Aktiivinen sovellustestaus parantaa sovelluksen toimivuutta, mikä taas parantaa sovelluksen laatua teknisestä näkökulmasta. Monialustainen sovellusratkaisu ei saa kuormittaa testausyksikköä liikaa, jotta sovelluksen laadunvarmistus testauksen toimesta ei kärsi.

Käyttöjärjestelmäalustojen erilaisuudet sekä niiden tarjoamat laitetuet voidaan myös nähdä haasteena sovelluksen laadun kannalta. Monialustaisen mobiilisovelluksen täytyy toimia kaikilla käyttöjärjestelmäalustoilla samalla tavalla sekä olla ulkoasultaan pääpiirteittäin samanlainen. Mikäli sovellus hajautuu erilaisiin toiminnallisuuksiin tai graafisiin ratkaisuihin, on kyseessä pahimmassa tapauksessa jo eri tuote eri alustoilla. Ideana Mobilogin kaltaisessa monialustaisessa mobiilisovelluksessa on tarjota käyttäjälle mahdollisimman yhteneväinen käyttökokemus, oli laite tai käyttöjärjestelmäalusta mikä tahansa. Mikäli sovellusta käyttävä asiakas joutuu valitsemaan tietyn laitteen sovelluksen ominaisuuksien takia, on laatumääritelmä virheellinen.

Jotta monialustainen mobiilisovellus pysyisi tasalaatuisena kaikilla alustaratkaisuilla, on kaikkia käännöksiä kehitettävä tasaisesti. Kehitystiimi joutuu pohtimaan ja selvittämään uuden ominaisuuden tuottamista sovellukseen huomattavasti tarkemmin kuin aiemmin, sillä jokaisen toiminnallisuuden suunnitteluvaiheessa täytyy olla selvillä eteen tulevat mahdolliset alustarajoitteet. Useamman alustan kehitystyö on huomattavasti työläämpää ja enemmän aikaa vievää verrattaessa yhden käyttöjärjestelmäalustan natiivin sovelluksen

kehitystyöhön. Suurempi työmäärä hidastaa vääjäämättä kehitystyötä, joka taas jarruttaa sovelluksen pysymistä teknologian huipulla. Sovelluksen laadun kannalta ajateltuna tekniikassa edelläkäyvät sovellukset lyövät helposti pois vanhahtavat tuotteet markkinoilta.

Mobiilikäyttöjärjestelmien tulevaisuuden näkymät asettavat myös haasteensa sovelluksen laadulle. Vaarana on, että yksi tuetuista alustaratkaisuista lakkautetaan kokonaan valmistajan toimesta. Tämä voi aiheuttaa sovelluksen kehityksen hiipumisen kyseiselle alustalle. Mikäli asiakas on valinnut käyttöönsä esimerkiksi vain yhden valmistajan laitteita, ei alustatuen hiipumisesta johtuva laitevaihdos ole välttämättä halpa ja helppo ratkaisu. Jotta sovellustoimittaja voi taata sovelluksen toiminnan jatkossakin, täytyy jokainen sovellusalustaratkaisu puntaroida tarkkaan ennen sen kehitystyön aloittamista. Turha alustakäännös syö resursseja kehitystiimiltä sekä vie uskottavuutta sovellusvalmistajalta, mikäli sovelluksen alustaratkaisu loppuu äkillisesti. Esimerkiksi Mobilog-sovelluksen kanssa suositellaan CGI:n toimesta validoituja päätelaitteita ja lopullinen laitevalinta tehdään aina yhdessä asiakkaan kanssa.

4 Laadunvarmistus Mobilog-monialustaratkaisussa

Tekniset seikat asettavat lopulta selkeän linjauksen Mobilog-monialustasovelluksen toteutukselle. Koska sovellus on toteutettu jo nyt *HTML5*-hybridi menetelmällä, sovelletaan valittua mallia sovelluksen monialustatoteutuksessa jatkossakin. Keskeisenä seikkana laadunvarmistuksen kannalta on testausmallien kehittäminen niin, että ne soveltuvat myös monialustasovelluksen laadunvarmistukseen.

4.1 Toteutusmenetelmänä HTML5-hybridisovellus

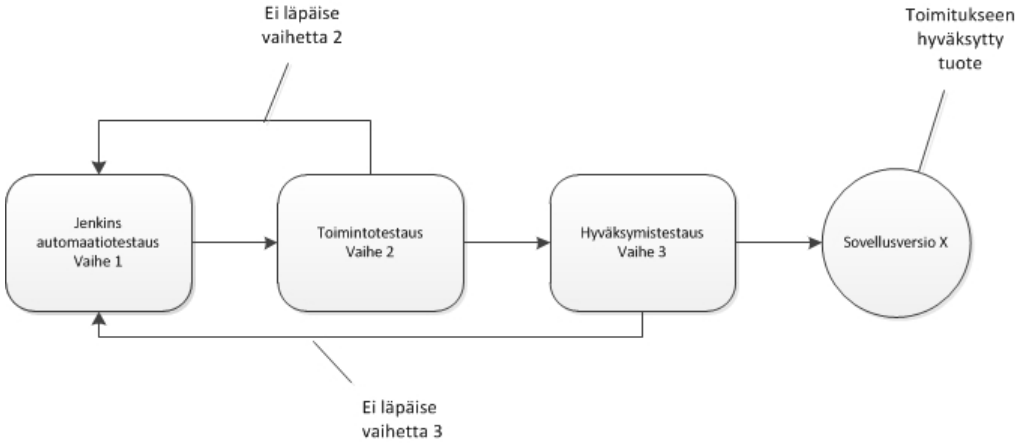
Mobilog-monialustasovelluksen toteuttaminen *HTML5*-hybridinä on luonnollinen ratkaisu, sillä sovelluksen toiminnassa keskeisessä roolissa ovat *kamera-* sekä *NFC*-toiminnallisuudet. Koska kyseiset ominaisuudet ovat todella heikosti toteutettavissa puhtaalla *HTML5*-teknologialla, on järkevin ratkaisu käyttää apuna käyttöjärjestelmäalustojen tarjoamia ohjelmistopaketointiratkaisuja. Hybridi *HTML5*-sovellustekniikka mahdollistaa sovelluksen muunnettavuuden puhtaaksi *HTML5*-sovellukseksi tulevaisuudessa, sillä ainoastaan *NFC-* sekä *kamera*-liitännäiset tehdään käyttäen erillistä koodiratkaisua. Esimerkkiratkaisuksi tehtävä WP8-käännös sovelluksesta sisältää *PhoneGap*-sovelluskehyksen sekä sitä tukevan pohjaprojektin. WP8-sovellusratkaisun arkkitehtuurin suunnittelu toteutetaan niin, että se ei aiheuta erillistä kuormaa testaukselle. Ratkaisussa täytyy pyrkiä hyödyntämään mahdollisimman paljon valmiita hyväksi todettuja ohjelmointiratkaisuja, kuten esimerkiksi *Gamman suunnittelumalleja* [12], sekä valmiita ohjelmistokirjastoja. Tällöin käännöksestä saadaan helpommin laadullisesti hyvä ja toimiva kokonaisuus, joka ei poikkea *HTML5*-toteutuksen osalta jo tällä hetkellä valmiista Android-käännöksestä. Puhtaan *HTML5*-sovelluksen toteuttaminen Mobilogin asettamien vaatimusten tasoon on vielä teknisesti haastava toteuttaa sekä epävarma toiminnaltaan puuttuvien ohjelmistorajapintojen takia.

4.2 Mobilog-testausmalli

Haastattelin testausosaston kollegaani Toni Kapiaista 14.3.2014, koskien käyttämäämme sovelluksen testausmallia sekä selvittääkseni mahdolliset haasteet monialustatestauksessa. Testaus on keskeinen elementti teknisessä laadunvarmistusprosessissa, sillä sen avulla voidaan varmistaa sovelluksen laatumääritteet sekä todentaa kirjoitetun ohjelmakoodin toimivuus. Tällä hetkellä Mobilog-sovellustestaus suoritetaan pääosin kahdessa osassa, jotka ovat *kehitetyn toiminnallisuuden testaus* sekä *hyväksymistestaus*. Lisäksi ennen testausosaston käsittelyä koodi hyväksytetään automaattisella yksikkötestauksella (*Jenkins*). Uuden toiminnallisuuden testauksessa keskitytään testaamaan sovellukseen kehitettyä ominaisuutta omana kokonaisuutenaan ja erillään muista toiminnallisuuksista. Menetelmä antaa kuvan kyseisen ominaisuuden toiminnasta sekä sen mahdollisista ongelmista. Hyväksymistestaus pyrkii löytämään Mobilog-sovellukseen kehitettyjen uusien ominaisuuksien asettamia haasteita ja ongelmia koko sovelluskokonaisuudelle. Hyväksymistestauksessa tuotteen kaikki parametrisoitavat ominaisuudet tulee kytkeä päälle, jotta mahdolliset ongelmakohdat ja ristiriidat muihin toiminnallisuuksiin nähden löytyisivät. Haastattelun tarkoituksena oli löytää testauksen läpivientiin toimiva malli, joka toimii myös usean alustaratkaisun toteutuksessa. Tärkeänä kriteerinä Toni Kapiainen totesi olevan monialustasovellusten toteutuksien samankaltaisuus toistensa kanssa (liite1). Tämä tarkoittaa sitä, että monialustainen tuote ei saa lisätä merkittävästi testausyksikön painotaakkaa ja testitapauksien täytyy pystyä olemaan yhdenmukaisia sovellusratkaisusta riippumatta.

4.3 Mobilog-testausprosessin kulku

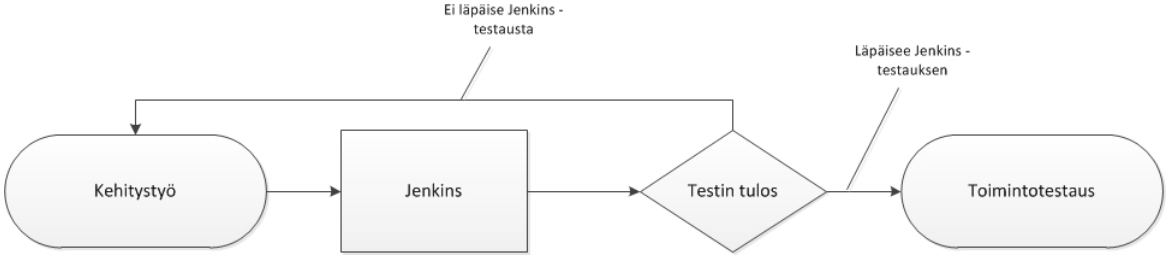
Haastattelun sekä nykyisin CGI:llä olevan testausmallin pohjalta voidaan muodostaa kuvassa 7 esitetty testausmalli, joka mahdollistaa Mobilog-mobiilisovelluksen testaamisen tehokkaasti. Mallin käyttö ei ole sidottu erityisesti mihinkään tiettyyn alustaan, tai sovellusversioon, joten se palvelee myös monialustaisen sovelluksen testausta.



Kuva 7: Laadunvarmistuksen kulku

4.3.1 Jenkins-automaatiotestit

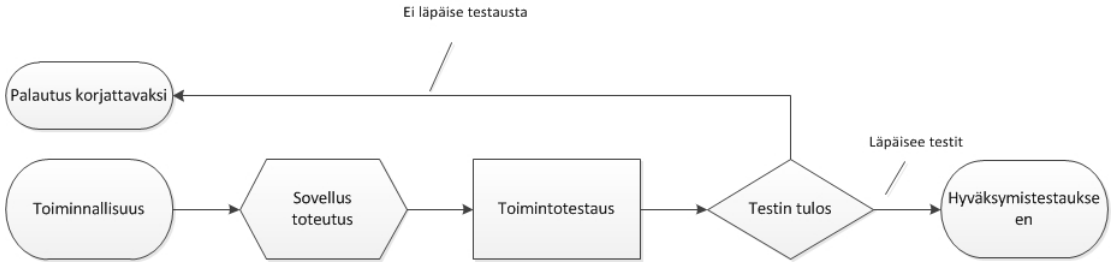
Ensimmäinen vaihe testausprosessissa on *Jenkinsin* suorittama automaattinen yksikkötestaus. Sovellukseen kehitetty uusi toiminnallisuus esitestataan kehittäjän koneella lokaalisti, jonka jälkeen kehitetty toiminnallisuus liitetään *GIT*-versionhallinnan *Jenkins*-testaukseen tarkoitettuun haaraan tai vaihtoehtoisesti kehityksessä käytettyyn *Toiminto*-haaraan. *Jenkins*-automaatiotestaussovellus suorittaa JavaSript–yksikkötestaukseen tarkoitetut *Jasmine*-yksikkötestit automaattisesti. Jotta sovelluksen uusi toiminnallisuus voidaan päästää testauksen seuraavaan vaiheeseen, on sen läpäistävä ensin automaatiotestaus yksikkötestitasolla. Mikäli *Jenkins* ei pysty suorittamaan testejä onnistuneesti läpi, palautetaan kehitetty toiminnallisuus takaisin sovelluskehittäjälle korjattavaksi. Vaihetta yksi toistetaan niin kauan, että kyseinen toiminnallisuus läpäisee automaattiset yksikkötestit. Monialustasovelluksessa jokaisen alustaratkaisun on mentävä läpi yksikkötesteistä, ennen kuin kehitetty toiminnallisuus voidaan päästää seuraavaan testivaiheeseen (liite1). Kuvassa 8 on esitetty Jenkins-testauksen läpikulkumalli.



Kuva 8: Jenkins-automaatiotestauksen kulku.

4.3.2 Toimintotestaus

Uuden toiminnallisuuden läpäistessä *Jenkins*-automaatiotestit voidaan se hyväksyä testausosaston käsittelyyn testauksen seuraavaan vaiheeseen. Kuten kuvasta 9 käy ilmi, toimintotestauksessa testataan kehitettyä toiminnallisuutta mahdollisimman irrallaan muista sovelluksessa olevista toiminnallisuuksista, jotta mahdolliset ongelmat tulevat paremmin esiin. Testaus suoritetaan käyttämällä ohjelmaa käsin ja samalla vertaamalla testattavaa toiminnallisuutta sprintissä määritettyyn käyttäjätarinaan. Käyttäjätarinassa on määritetty ne hyväksymiskriteerit, jotka jokaisen monialustasovelluksen tulee täyttää, jotta se voidaan hyväksyä osaksi lopullista sovelluskokonaisuutta. Mikäli yksikin testattava monialustaratkaisun toiminnallisuus ei täytä sille asetettuja hyväksymiskriteereitä, tai siinä on sovelluksen toiminnan kannalta ongelmia, palautetaan se takaisin sovelluskehittäjille. Korjattua toiminnallisuutta ei palauteta takaisin *Toimintotestaukseen*, vaan sen täytyy kulkea uudestaan sekä *Jenkins*–automaatiotestauksen että *Toimintotestauksen* kautta, jotta se voidaan päästää eteenpäin testausketjussa. Menettelyllä pyritään varmistamaan, ettei toiminnallisuuteen tehdyt korjaukset ole vaikuttaneet yksikkötestien toimivuuteen.

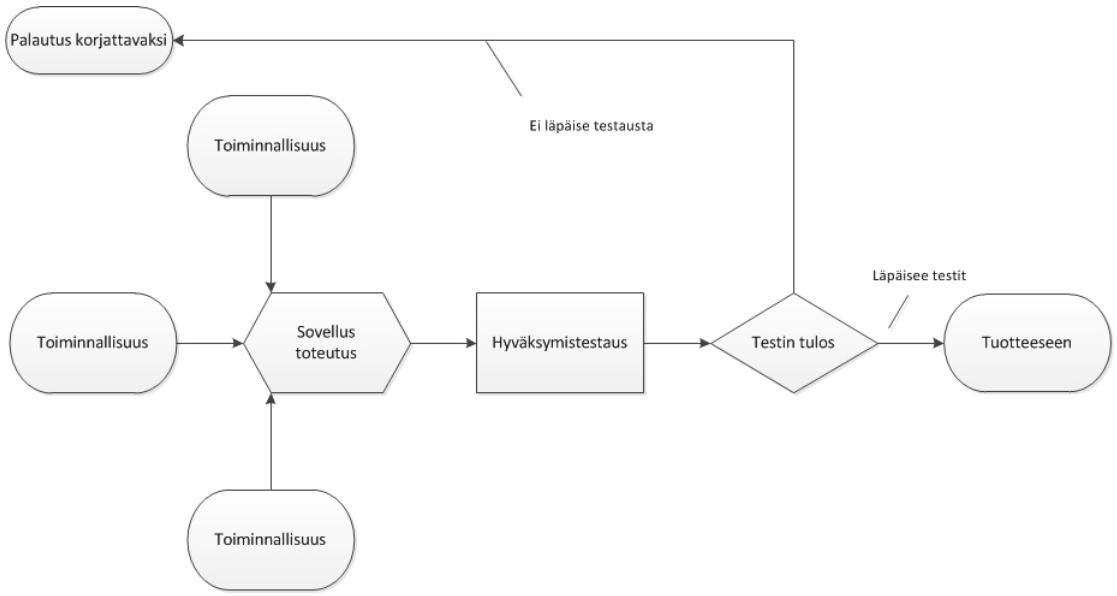


Kuva 9: Toimintotestauksen kulku.

4.3.3 Hyväksymistestaus

Hyväksymistestaukseen päästyään kehitetty toiminnallisuus liitetään osaksi koko mobiilisovellusta, jolloin siitä pyritään löytämään sen mahdollisesti muille ominaisuuksille tai muiden ominaisuuksien sille aiheuttamat toimintahäiriöt. Tällöin on erittäin tärkeää, että sovelluksen kaikki parametrisoitavissa olevat ominaisuudet ja toiminnallisuudet on kytketty päälle. Testauksen kulku tapahtuu suorittamalla kaikki toiminnallisuuden käyttäjätarinan vaatimukset uudestaan, sekä myös kaikki mahdolliset käyttökombinaatiot, jotka voisivat aiheuttaa virheen sovelluksen toiminnassa. Mikäli testattava monialustasovellus ei toimi kokonaisuutena oikein, palautetaan hylkäyksen aiheuttanut toiminnallisuus jälleen

takaisin sovelluskehittäjälle korjattavaksi kuvan 10 osoittamalla tavalla. Mikäli hylkäys koskee vain yhtä monialustaratkaisua, joutuvat muut läpi menneet ratkaisut odottamaan toiminnallisuuden lopullista hyväksymistä niin kauan, että kaikki monialustatoteutukset läpäisevät koko testiketjun. Kun toiminnallisuuden testaus on hyväksytysti suoritettu, pääsee testattavana ollut uusi monialustatoiminnallisuus sprintin lopussa valmistuvaan sovellusversioon mukaan.



Kuva 10: Hyväksymistestauksen kulku.

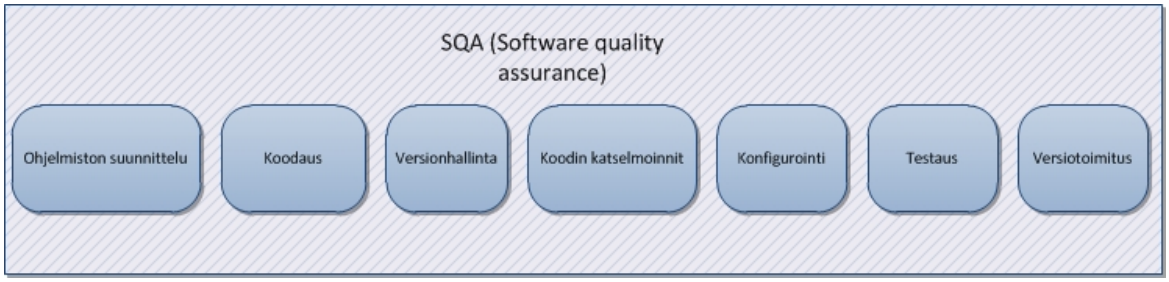
4.4 Monialustasovelluksen sovittaminen kolmivaiheiseen testausmalliin

Pelkästään jo yhden sovellusratkaisun tehokas testaus on aikaa vievää työtä, joten manuaalisen työn vähentäminen kolmivaiheisessa testausketjussa on tärkeää. Monialustaisen mobiilisovelluksen kohdalla, testaaja joutuu käymään läpi kaikki mahdolliset testauskombinaatiot jokaisella sovellusalustalla joten työresurssit kasvavat helposti liian suuriksi. Koska monialustatestauksessa kaikkien testattavien sovellusversioiden on mentävä läpi jokaisesta testausvaiheesta ennen kuin sprintistä tuleva versio voidaan hyväksyä toimitettavaksi, täytyy toteutuksien olla toiminnaltaan ja arkkitehtuuriltaan mahdollisimman yhtenevät. Yhdenmukainen arkkitehtuuri mahdollistaa testauksen kuorman pysymisen tasaisena, jolloin sovelluksen laatu ei kärsi kiireen takia. Hybridi *HTML5*-sovellus sisältää kaikille alustaratkaisuille sovelluksen yhteneväisen toiminnallisen logiikan sekä käyttöliittymän, jolloin testauksen kehittäminen automatisoituun suuntaan on mahdollista. Automaattinen sovellustestaus mahdollistaa sovelluksen toiminnallisten logiikoiden testauk-

sen tietokoneen avulla automatisoidusti, jolloin testaajan ei tarvitse tehdä välttämättä *toimintotestausta* käsin alusta kerrallaan. Automaattisen sovellustestauksen käyttöönotto voidaan nähdä edellytyksenä monialustatestauksessa, jotta laadunvarmistus testauksen osalta ei kärsi sovelluksessa. Kolmivaiheinen testausmallin toimivuus edellyttää jokaisen monialustasovelluksen virheetöntä läpäisyä testauksesta, eikä poikkeuksia voi sallia. Mikäli joku alustaratkaisu ei täytä kriteereitä ja se jää muista kehityksessä jälkeen, on koko monialustasovelluksen laatu kyseenalaistettavissa.

4.5 Laadun mittarit Mobilog-monialustasovelluksessa

Jotta sovelluksen laatua voidaan seurata ja mahdollisiin laatuongelmiin puuttua, täytyy määrittää mittarit, joiden pohjalta soveluksen laatu voidaan todentaa. Sovelluksen laadun varmistus eli *SQA (Software quality assurance)* auttaa määrittelemään sovelluksen suunnittelussa laadun kannalta hyväksi havaittuja menetelmiä. Laatumääritelmät voivat nojata yleisiin laatustandardeihin kuten *ISO9000*, *CMMI* [13], tai sovellusta kehittävä taho voi määritellä omat laatukriteerinsä tuotteelle. Kuvassa 11 esitetty *SQA* kattaa kokonaisuudessaan koko sovelluksen kehitysprosessin, kuten ohjelmiston suunnittelun, ohjelmoinnin, katselmoinnit, konfiguroinnit, testauksen sekä julkaisun [14]. Tarkoituksena on pystyä sovittujen menetelmien avulla varmistamaan laatuprosessin oikeanlainen kulku. Seuraavissa kappaleissa pyritään löytämään keskeisiä menetelmiä määrittämään laadun mittareita Mobilog-monialustasovellukseen sekä tekniikoita, jotka helpottavat monialustaisen mobiilisovelluksen kehitystyötä.



Kuva 11: Laadunvarmistuksen kohteet

4.5.1 Testauksen keskittäminen asiakkaan tarpeisiin

Sovelluksen laadun kannalta on elintärkeää keskittyä testaamaan niitä ominaisuuksia, joita asiakkaat eniten tuotteessa käyttävät. Tuotteen perusominaisuuksien on toimittava asiakkaan kanssa luotujen määritteiden kuvaamalla tavalla, joten tiivis yhteistyö asiakkaiden kanssa on ensiarvoisen tärkeää. Tehokas yhteistyö takaa sen, että kehitettävä toiminnallisuus on asiakkaan toiveiden mukainen. Asiakkaan kanssa tehtävä tiivis yhteistyö on osa ketterää Scrum-kehitysmallia [15], mutta yleisesti yhteistyötä tehdään aina vasta sprintin päättyessä. Varsinkin monialustasovelluksessa yhteistyö asiakkaiden kanssa on tärkeää, sillä sovellusta koskeviin muutoksiin reagoiminen on hitaampaa johtuen suuremmasta painolastista kehitys ja testaus -yksiköissä. Menetelmän kannattavuutta voidaan mitata vuosittain tehtävillä asiakastyytyväisyysmittauksilla, joiden tulokset kootaan yhteen ja analysoidaan Scrum-palavereissa.

4.5.2 Koodin lausekattavuus

Mikäli monialustaratkaisun pohja on toteutettu samankaltaisilla tekniikoilla, voidaan sovelluksen oikea toiminta varmistaa helpommin kuin hajanaisessa ratkaisussa. Yksikkötestien sekä arkkitehtuuriratkaisuiden täytyy olla yhdenmukaiset jokaisessa alustaratkaisussa, jotta testitapaukset tuottavat luotettavaa mittausdataa testattavasta toiminnallisuudesta. Mikäli kehitys hajoaa eri sovellusratkaisuiden kohdalla toisistaan paljon poikkeaviin toteutuksiin, on testauksen täysin mahdotonta saada luotettavaa tietoa monialustasovelluksen toiminnasta. Heikko arkkitehtuurisuunnittelu johtaa vääjäämättä eri alustaversioiden hajaantumiseen ja pahimmillaan toiminnallisuuksien erovaisuuksiin eri alustaratkaisuissa. Testien toiminnan samankaltaisuutta voidaan mitata esimerkiksi koodin lausekattavuuden avulla, joka on osa koko *koodin kattavuus* (Code Coverage) -laadunvarmistus menetelmää. Koodin kattavuus voidaan jakaa karkeasti neljään osaan: *funktioiden kattavuuteen*, *lauseiden kattavuuteen*, *kontrollirakenteiden kattavuuteen*, sekä *tilojen kattavuuteen* [16]. Menetelmän tarkoituksena on varmistaa, että mahdollisimman moni koodirivi suoritetaan testien toimesta, jotta mahdolliset virheet voidaan havaita[14]. Monialustaisessa sovelluksessa eri alustojen lausekattavuuksia tulee vertailla keskenään ja näin ollen varmistaa, että jokainen sovellusratkaisu on yksikkötestien puolesta yhtenevällä tasolla muiden alustaratkaisuiden kanssa.

4.5.3 Virheindeksit

Virheindeksi soveltuu monialustasovelluksen teknisen laadun mittaamiseen. Menetelmän ideana on, että jokaisen versiojulkaisun pohjalta kerätään tietoa sovelluksesta löytyneiden virheiden kokonaismäärästä, sekä siitä, mitkä toiminnallisuudet ovat virheitä aiheuttaneet (liite1). Keräämällä tietoa sovelluksessa ilmenneistä ongelmista saadaan aikaan tietovarasto, josta voidaan poimia mahdollsia ongelmakohtia sovelluksen kehityksessä. Kun ongelmien aiheuttajat on selvitetty, voidaan niihin varautua paremmin jatkossa, jolloin esiintyvien virheiden määrän pitäisi kääntyä laskuun. Indeksin toiminta edellyttää saumatonta yhteistyötä kehitys- ja testausyksikön välillä, jotta tieto esiintyneistä virheistä tavoittaa sovelluskehittäjät mahdollisimman nopeasti. Tällöin sovelluksesta löytyneen virheen syyt voidaan käydä läpi nopeammin kehitysyksikön kanssa ja samalla oppia välttämään vastaavanlaisia ongelmatilanteita jatkossa. Optimaalisena tavoitteena on sovelluksen täydellinen virheettömyys, mutta se on mahdollista vain teoreettisesti. Menetelmässä tulee asettaa sovellukselle mielekäs virheprosentti, jonka alittamiseen sovelluksen kaikilla osa-alueilla pyritään. Tietoa ilmenneistä virheistä tulee kerätä esimerkiksi vuoden ajalta ja virheet tulee lajitella niiden tyyppien mukaisesti (liite1). Indeksien lopputulokset tulee käsitellä aina *sprintin jälkiarviointi tilaisuudessa*, jossa käsitellään kehitysjakson aikana ilmenneitä onnistumisia ja epäonnistumisia. Indekseistä saadaan yksittäisiä kohteita ja osa-alueita, joissa kehityksen kulkua voidaan parantaa jatkossa. Vuoden lopussa virheindeksit kootaan yhdeksi dokumentiksi, jolloin nähdään kokonaiskuva vuoden aikana tulleista ongelmista sekä niihin reagoimisesta. Menetelmän avulla ongelmat voidaan paikantaa tarvittaessa alustakohtaisesti, mikä helpottaa monialustaisen sovelluksen laadun tarkkailua.

4.5.4 Käyttötestit

Osana hyväksymistestausta suoritetaan sovelluksen käyttötesti noudattaen ennalta luotua Excel-taulukkoa. Taulukossa on määritetty kohta kohdalta erilaiset käyttötilanteet ja tapaukset, joista monialustasovelluksen tulee suoriutua. Testaajan tulee arvioida, täyttääkö hyväksymistestauksessa oleva sovellus toiminnallisuuksien kuvausten mukaiset määritelmät ja tehdä sen pohjalta päätös, voidaanko käyttötesti hyväksyä. Käyttötestitaulukkoon tulee merkitä jokaisen tapauksen kohdalle OK-merkintä, mikäli testattava toiminnallisuus täyttää sille asetetut hyväksymiskriteerit.

Käyttötestitaulukoista voidaaan tarkastaa toiminnallisuuden aikaisempi toiminta, mikäli uudessa versioinnissa johonkin vanhempaan toiminnallisuuten on ilmaantunut virheitä. Dokumentissa tulee olla tehdyn käyttötestin mittauspäivämäärä, jotta toiminnallisuuksien tarkastaminen ja ongelmien paikantaminen on mahdollista. Käyttötestin automatisointi on hankalaa, sillä sovellusta täytyy testata oikeissa *käyttöolosuhteissa* (kenttäolosuhteissa). Monialustainen sovellusratkaisu asettaa haasteita lähes jokaisella sovellusalueella, joten perusteellinen käyttötestaus kaikilla sovellusratkaisulla on välttämätöntä. Käyttötestidokumenttien vertailu antaa hyvän pohjan varmistaa sovelluksen oikeanlainen toiminta jokaisen *sprintin* pääteeksi, sekä niistä saatu tieto voidaan yhdistää virheindeksien kanssa. Koska virheindeksi on enemmän teknologiapohjainen ratkaisu, täydentää käyttötestidokumentaatio sitä koko sovelluksen kattavaksi laatudokumentiksi.

5 Ratkaisuja monialustasovelluksen laadunvarmistukseen

Haastattelun ja edellä esiteltyjen menetelmien pohjalta esitän parhaaksi näkemiäni kehitysideoita ja toimintamalleja, jotka ennaltaehkäisevät virheitä sekä parantavat monialustasovelluksen laatua. Menetelmät pätevät yleisesti sovelluskehitykseen, joten niiden käyttö muussakin kuin mobiilikehityksessä on mahdollista. Osa ratkaisumalleista on jo osittain Mobilog-sovelluksen kehitystyössä käytössä, mutta tässä työssä niitä on tarkennettu ja jatkokehitetty tukemaan paremmin tulevaa monialustakehitystä.

5.1 Hyväksymistestauksen kehittäminen

Hyväksymistestauksen yhteydessä jokaista monialustaratkaisua tulisi testata erityisesti sellaisilla käyttäjillä, jotka eivät kuulu suoraan sovelluksen kehitystiimiin. Menetelmä tarjoaa sellaista käyttäjätietoa, mikä jää helposti sovelluksen jo tuntevalla testaajalla pimentoon. Suoritetun käyttötestin jälkeen valikoidun testihenkilön tulee täyttää hänelle annetulle kaavakkeelle tekemänsä havainnot ja mahdolliset puutteet sovelluksessa. Kehitystiimin ulkopuolisen testaajan havainnot tarjoavat realistisemman kuvan sovelluksen todellisesta toiminnasta, sillä testikäyttäjä ei ota kantaa sovelluksen tekniseen toteutukseen. Menetelmä voidaan toteuttaa joko valikoitujen asiakkaiden kanssa, tai sovellusta voidaan vaihtoehtoisesti testauttaa yrityksen sisällä eri yksiköissä. Testihenkilöiden tulee vaihtua aika ajoin, jotta tulokset pysyvät vertailukelpoisina tulevaisuudessa. Pitkään testihenkilönä ollut käyttäjä voi alkaa tuotteen paremmin tuntiessaan ''sokeutua'' sovelluksen mahdollisille loogisille virheille. Hyväksymistestauksen yhteydessä tulee ottaa käyttöön laadun mittareista saatava tieto (virheindeksit ja käyttötestit). Tiedon yhdistäminen laadun mittareiden, hyväksymistestauksen havaintojen sekä ulkopuolisen testaajan raporttien pohjalta, luo yhdessä kattavan faktaan perustuvan dokumentaation sovelluksen yleisestä laadusta.

5.2 Suunnittelumallien käyttöönotto ohelmistosuunnittelussa

Erich Gamma käsittelee kirjassaan *Design Patterns: Elements of Reusable Object-Oriented Software* [12] suunnittelumallien käyttöä sovelluskehityksen apuvälineenä. Suunnittelumallien päätavoitteena on tarjota valmiita loogisia toimintamalleja yleisiin ohjelmoinnissa esiintyviin ongelmatilanteisiin. Mallit on esitetty alun perin *C++* -kielisinä ratkaisuina, mutta ne toimivat myös yleisesti lähes kaikissa olio-ohjelmointikielissä. Mobilog–sovellus on kirjoitettu pelkästään käyttäen oliopohjaisia ohjelmointikieliä, mikä mahdollistaa Gamman määrittelemien mallien käyttöönoton sovelluksen suunnittelussa ja toteutuksessa. Tämänhetkisessä HTML5-toteutuksessa malleja ei ole otettu käyttöön tietoisesti, joten jatkoa ajatellen mallien käyttöä voi uusissa toiminnallisuuksissa kokeilla. Suunnittelumallit, tai yleisesti tunnettujen arkkitehtuurien käyttö takaa koodin yhtäläisen testattavuuden monialustaratkaisuissa, sekä ratkaisujen tehokkaan siirtämisen eri alustojen välillä. Tarjolla olevat ratkaisumallit helpottavat merkittävästi suunnittelu- ja toteutustyötä, sillä kaikilla sovellukseen osallistuvilla kehittäjillä on tarkoin määritelty yhteinen linja kirjoitettaessa koodia sovellukseen. Suunnittelumallit ovat valmiiksi testattuja ongelmanratkaisumenetelmiä, joita toteuttamalla loogisten virheiden määrä koodissa laskee. Mallien käyttö parantaa sovelluksen laatua, sillä sovelluskehittäjien omat virhealttiit ratkaisut jäävät pois sovelluskoodista. Suunnittelumallien yhteydessä on hyvä käyttää myös tunnettuja ja testattuja valmiita ohjelmakirjastoja, jotka löytyvät eri ohjelmointikielistä. Kirjastojen tehokas käyttö ehkäisee koodivirheiden syntymistä ja pitää sovelluksen koodin helpommin luettavana riippumatta kielestä. Kokonaisuutena suunnittelumallit ja tehokas ohjelmakirjastojen käyttö ohjaavat koko sovelluksen kehitystyötä sekä yhdenmukaistavat sovelluksen ohjelmakoodia sekä sen toteutusta.

5.3 TDD ja parikehitys

Mobilog-projektissa olevien kehittäjien tulee sitoutua *TDD (Test Driven Development)* -kehitysmalliin, jossa yksikkötestit kirjoitetaan ennen varsinaista ohjelmakoodia. Menetelmän ideana on kirjoittaa aina uutta toiminnallisuutta vastaava testitapaus ja muokata koodi toimimaan siten, että se toteuttaa alkuperäisen testitapauksen [17]. Monialustaisen sovelluksen kehityksessä varsinkin testitapaukset nousevat suureen arvoon, sillä testit on suunniteltava siten, että jokainen alustakäännös tukee suunniteltua toiminnallisuut-

ta. Testien avulla toiminnallisuutta voidaan simuloida jokaisella alustaratkaisulla ja luoda raamit toiminnallisuuden toteutukselle. Testivetoinen kehitysmalli on järkevintä toteuttaa esimerkiksi *pariohjelmoinnin* avulla (XP / Extreme Programming). Pariohjelmointi perustuu kahden kehittäjän väliseen yhteistyöhön, jossa he kehittävät samaa sovellusosaa yhtä aikaa [18]. Aluksi *TDD:n* ja *XP:n* yhdistelmässä molemmat kehittäjät pyrkivät määrittelemään kattavat testitapaukset kehitettävän toiminnallisuuden kannalta. Itse koodin kehityksen alkaessa, voi toinen kehittäjä alkaa laahjentamaan testien kattavuutta ja kehittää testejä pidemmälle. Toisen osapuolen tehtävänä on kirjoittaa koodi, joka vastaa testien määrittelyä. *TDD/XP* -menetelmissä testit ohjaavat koodin kirjoitusta, joka johtaa välttämättä parempaan koodin lausekattavuuteen sekä toimintaan. Väite voidaan perustella siten, että testitapaukset täyttävä ohjelmakoodi on suunniteltu ja toteutettu kahteen kertaan eri kehittäjien toimesta. Toimiviksi havaittujen testitapausten jakaminen muiden parikehitystiimien välillä on suotavaa, jotta toimivien testitapausten kierrättäminen tehostuu koodissa. Tällöin vältytään saman asian kirjoittamiselta useampaan kertaan.

5.4 Pyrkimys yhteen ohjelmistoarkkitehtuuriin

Monialustaisen Mobilog-sovelluksen kehityksessä on kannattavaa pyrkiä jatkoa ajatellen mahdollisimman paljon yhteen arkkitehtuuriratkaisuun. Kuten jo aiemmin työssä totesin, tämä ei ole vielä *HTML5*-teknologian kanssa täysin mahdollista, mutta alustakohtainen pakettikäärintä ei myöskään vie kehitystä väärään suuntaan. Tämä on perusteltavissa sillä, että sovelluspaketointi tarjoaa ainoastaan *NFC-*, *kamera-* ja *GPS* -tuen sovellukselle. Muu sovelluslogiikka sijaitsee edelleen *HTML5*-koodissa, joten toiminnallisuuksien siirtäminen alustalta toiselle on jatkossakin helppoa. Hybridi *HTML5*-sovellus on hyvä ja joustava ratkaisu kehittää sovellusta eteenpäin, mutta toiminnallisen logiikan kirjoittamista muualle kuin *HTML5*-koodiin on vältettävä. Mikäli sovelluksen toiminnallinen koodi hajautuu usealle eri ohjelmointitekniikalle, on sovelluksen siirtäminen alustalta toiselle huomattavasti monimutkaisempaa. Nykyisen *HTML5*-sovelluksen kehitystä tulee viedä eteenpäin ja korvata ainoastaan puuttuvia laitekohtaisia ominaisuuksia tilapäisesti sovelluspaketointiratkaisuilla. Lopulta kattavan *HMTL5*-laitetuen valmistuessa, voidaan luopua alustakohtaisista sovellusratkaisuista ja siirtyä käyttämään pelkästään *HTML5*-pohjaista tekniikkaa. Hybridisovellusratkaisu toimii lopulta eräänlaisena siirtymätienä puhtaaseen HTML5-sovellusratkaisuun.

5.5 Mobiilisovelluksen automatisoitu käyttötestaus

Automaattiseen käyttöliittymätestaukseen on tehty paljon työkaluja, joilla osa sovelluksen käyttötestauksesta voidaan suorittaa ilman testaajan aktiivista työskentelyä. Automaattisen käyttölittymätestauksen ideana on ohjelmoida halutut käyttötestitapaukset valmiiksi tietokoneelle, joka sen jälkeen suorittaa kaikki mahdolliset käyttötapauskombinaatiot sovelluksessa (liite1). Käyttötapauskombinaatioita muodostuu yksinkeraisessakin sovelluksessa paljon ja kaikkien testipolkujen käyminen läpi käsin on hidasta ja virhearkaa. Mobiilin monialustasovelluksen kohdalla ongelmaksi muodostuvat erilaiset päätelaitteet. Koska automaattinen käyttöliittymätestaus voidaan toteuttaa ainoastaan tietokoneella toimivan sovelluksen kautta, ei testattavaa käännöstä voida käyttää fyysisesti sille tarkoitetulla laitteella. Ongelmaan yhtenä ratkaisuna on käyttää jokaisen alustakäännöksen kohdalla alustakohtaista emulaattoria, jolla voidaan suorittaa testattavaa sovellusta tietokoneella. Emulaattori ei tarjoa aivan täydellistä toiminnallista samankaltaisuutta kuin oikea päätelaite, mutta sen avulla voidaan varmistaa jokaisen sovelluskäännöksen käyttöliittymäkohtainen looginen toimivuus. Käytännössä manuaaliseksi työksi jää ainoastaan *NFC*-luennan ja kameran toiminnallisuuden testaaminen sovelluksessa. Testaajalle jäävä laitekohtainen fyysinen testaus on huomattavasti pienempi operaatio, kun jokaisen sovellusratkaisun looginen testaus on suoritettu automaattisesti tietokoneen toimesta.

6 Mobilog WP8 -hybridisovellus

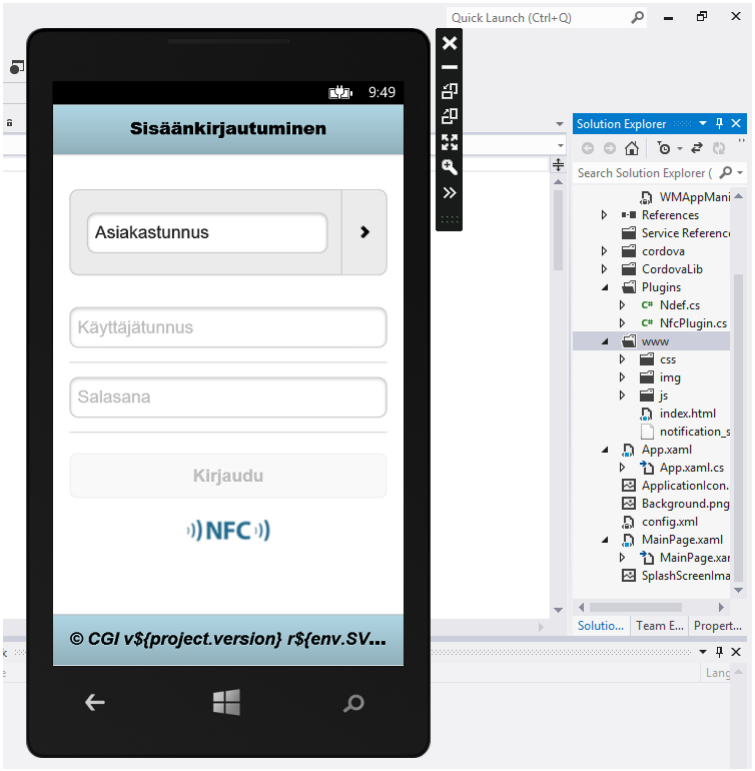
Monialustasovelluksen toteutustekniikaksi valikoitui lopulta kolmesta aiemmin esitellystä sovellusratkaisusta *HTML5*-hybridisovellus. Tarkoituksena on luoda jo aiemmin toteutetusta Mobilog-Android-sovelluksesta uusi sovelluskäännös, joka toimii Microsoft Windows 8 -mobiilikäyttöjärjestelmän päällä. Toteutuksen tulee läpäistä aiemmin esitetty testausmalli, sekä laadun kannalta oleelliset seikat. Kuten jo aiemmissa luvuissa totesin, käännöksessä ei tarvitse ottaa kantaa *HTML5*-koodin toteutukseen, sillä se toimii sellaisenaan jo nyt *WP8*-alustalla. Keskeiseksi työtehtäväksi jää sovelluspaketoinnin luominen *WP8*-alustalle sekä selvitystyö *NFC*-liitännäisen toimivasta toteutuksesta sovellusversioon.

6.1 Sovelluspaketoinnissa käytettävät tekniikat

Toteutuksen pohjana käytetään valmista *PhoneGap WP8* -pohjapakettia, joka sisältää sovelluspaketoinnin toteutukseen tarvittavat liitännäiset. Kehitystyö toteutetaan *Visual Studio 2012* -sovelluskehittimellä sekä Microsoftin yleisesti tarjoamilla kirjastoratkaisuilla. NFC-liitännäisen perustana käytetään valmista *NFCPlugin.cs*-luokkaa, jota voidaan muokataan halutun toiminnallisuuden saavuttamiseksi. Sovelluspaketoinnin toteutuskielenä käytetään *Microsoftin C#* -ohjelmointikieltä, joka on yhteensopiva *WP8*-alustan kanssa. Syy kyseisen kielen valintaan tulee *C#*-ohjelmoinnin aiemmasta tuntemuksesta, sekä *NFC*-liittännäisen *C#:lla* luodusta valmiista pohjatoteutuksesta.

6.1.1 Mobilog-sovelluskäännöksen luominen

WP8-sovelluksen pohjaratkaisu luodaan purkamalla ladattu PhoneGap-aloituspaketti Visual Studion projektit -hakemistoon tietokoneelle. Aloituspaketti sisältää valmiiksi *HTML5*-yhteensopivan hakemistorakenteen, minne valmiit Mobilog *HTML5* -lähdekoodit asetetaan. Mobilog-sovelluksen lähdekoodit kopioidaan luodun Visual Studio-projektin www-hakemistoon, kuten kuvassa 12 on esitetty.



Kuva 12: WP8-sovelluksen pohja.

HTML5-hybridisovellus on jo tässä vaiheessa käyttövalmis ja sen toimivuutta voidaan testata käyttämällä Visual Studion tarjoaamaa *WP8*-emulaattoria, tai *WP8*-puhelinta. Huomioitavaa kuitenkin on, että NFC-luenta ei vielä toimi tässä vaiheessa käännöstä.

6.1.2 NFC-liitännäisen toteuttaminen sovellukseen

NFC-luennan toteutuksen pohjana *WP8*-käännöksessä käytetään *Microsoft Developer Reference* [19] -ohjeistuksen mukaista *NDEF* (NFC Data Exchange Format) -mallin toteuttavaa *C#*-luokkaa. Luokka tarjoaa valmiit toteutukset funktioille, joiden avulla erilaiset *NFC*-toiminnallisuudet voidaan toteuttaa. Valmiin NDEF *C#* -luokan voi hakea Microsoft Visual studion *NuGet*-pakettienhallinan avulla, jolloin se asentuu autotomaattisesti oikeaan hakemistoon projektissa. NDEF-määritysluokan lisäksi tarvitaan itse NFC-liitännäinen, joka välittää NFC-piirin lukemaa tietoa PhoneGapin ja WP8-alustan välillä. Pohjana *NFC*-toteutukselle on järkevintä käyttää ennalta testattuja ratkaisuja, kuten *Nokia Developer -oppaan* [20] tarjoamia esimerkkejä, tai valmista*NFC*-toteutusta *WP8*-käyttöjärjestelmään.

Päädyin käyttämään valmista NFCPlugin.cs -luokkaa, joka on toteutettu CGI:ssä koostamalla useita eri luentaratkaisuja yhteen tiedostoon. NFC:n NDEF-alustus toteutetaan suoraan NFCPlugin.cs-luokassa, kuten kuvassa 13 on esitetty.

```
public void init(string args)
{
    // not used for WP8
    Debug.WriteLine("Init");
}

// no args
public void registerNdef(string args)
{
    Debug.WriteLine("Registering for NDEF");
    proximityDevice = ProximityDevice.GetDefault();
    subscribedMessageId = proximityDevice.SubscribeForMessage("NDEF", MessageReceivedHandler);
    DispatchCommandResult();
}

// no args
public void removeNdef(string args)
{
    Debug.WriteLine("Removing NDEF");
    if (subscribedMessageId != -1)
    {
        proximityDevice.StopSubscribingForMessage(subscribedMessageId);
        subscribedMessageId = -1;
    }
    DispatchCommandResult();
}
```

Kuva 13: NFC-liitännäisen alustus (NFCPlugin.cs).

Luokassa on toteutettu *NFC*-luentaan ja kirjoitukseen tarkoitetut C#-funktiot, joiden avulla tieto voidaan välittää PhoneGapin kautta *HTML5*-sovellukseen. PhoneGap konfiguroidaan kutsumaan NFCPlugin-luokkaa sovelluksen käynnistyessä, jonka jälkeen toiminnallisuus on valmis testattavaksi. Valmiiksi kirjoitettua *NFC*-luokkaa on hyvä käyttää projektin pohjana, sillä sen liittäminen koodiin on helppoa ja sen toimivuus on valmiiksi testattu. C# ei myöskään tarjoa juurikaan vaihtoehtoja tehdä tunnisteen luentaa omilla menetelmillä, sillä *WP8:n NFC*-toiminnallisuus pohjautuu valmiiden sovelluskirjastojen käyttöön.

Sovelluksen NFC-toteutus on konfiguroinnin jälkeen valmis ja luentaa voidaan testata puhelimella. Ongelmaksi ilmeni kuitenkin sovellusta testattaessa NFC-tunnisteen *UID*-tiedon lukeminen WP8-puhelimessa. Microsoft ei tarjoa rajapintaa, tai kirjastotoimintoja NFC:n UID:n luentaan, mikä on tehdyn sovelluskäännöksen kannalta huono asia. NFC *UID (Unique ID)* on matalan tason *NFC*-tunnisteen uniikki ID, jonka käyttö on osa Mobilogin nykyistä käyttäjän tunnistautumisen toteutusta. Periaatteessa pelkällä *NFC*-tietoalueen luennalla työsuoritekirjauksia voidaan suorittaa nykyisessä Mobilog-tuotteessa, mutta vanhemmissa työnohjaussovellustoteutuksissa *UID* on edelleen pakollinen.

6.1.3 WP8-sovelluskäännöksen lopputulos

WP8-sovelluskäännös ilman puhelimen lisälaitteiston tukea on yksinkertaista toteuttaa ja sovelluksen toiminnallisuudet vastaavat täysin Android-käännöstä sovelluksesta. Ongelmaksi muodostui Microsoftin tarjoaman *NFC*-rajapinnan puutteet luettaessa *NFC*-tunnistetta. Mobilog-järjestelmän toiminnan kannalta on olennaista, että NFC:n kautta voidaan lukea tunnisteen *UID* ja tietoalue. Periaatteessa sovellusta voidaan käyttää lukemalla pelkästään data-alue, mutta silloin käännös poikkeaa Android-käännöksen toiminnasta, koska osa toiminnallisuuksista jää puuttumaan. Toiminnallisuuden puute aiheuttaa ristiriidan aiemmin määrittelmissäni testauksen ja laadun mittareissa, sillä monialustaisen tuotteen tulee olla yhdenmukainen jokaisella järjestelmäalustalla. Tällöin sovelluksen onnistunut käännös tarkoittaisi UID:n poistamista koko työnohjaussovelluksen toiminnasta, jotta monialustakäännökset olisivat vertailukelpoisia.

Ohjelmapaketointiratkaisun tuottamiseen on käytetty ainoastaan *C#*-ohjelmointikieleen tehtyjä valmiita *NFC*-kirjastoja, joten laadullisesti *NFC*-toiminnallisuuden toteutus täyttää aimmin määrittelemäni tarpeet ohjelmakirjastojen tehokkaasta käytöstä sovelluksen toteutuksessa. Koska *HTML5*-koodi oli jo toteutusvaiheessa valmis ja testattu, ei sen arkkitehtuurimuutosta *Gamman suunnittelumalleihin* voi viedä eteenpäin ennen toteutusmallien tarkempaa tutkimista sovelluksessa. Jatkossa malleja voi ottaa käyttöön hiljalleen, jolloin niiden toimintaa voidaan verrata nykyiseen toteutustapaan.

WP8-hybridisovelluksen *HTML5*-yksikkötestit läpäisevät automaatiotestauksen yhdenmukaisesti Android-sovelluksen kanssa, joten sovelluskäännös ei vahingoita HTML5-toteutusta millään tavalla. Emme pystyneet suorittamaan kunnollista käyttötestiä *NFC*- ja *kamera*-puutteiden takia, joten täyttä varmuutta sovelluksen luotettavasta toiminnasta ei voi vielä antaa. Tehty sovelluskäännös on kuitenkin lupaava alku WP8-puhelinten ottamiseksi osaksi Mobilog-kokonaisuutta ja selvitämme parhaillaan, tarvitaanko *NFC* -tunnisteen *UID*-osuutta jatkossa sovelluskokonaisuudessa.

Sovelluksen laadun kannalta tarkasteltuna WP8-sovelluskäännös on täysin identtinen nykyisen Android-sovelluksen kanssa, joten nykyiset laatumääritteet tekniseltä osalta täyttyvät. Työssä esittelemäni laatumittarit ja parannukset ovat nyt esitysasteella Mobilog-kehitysprojektissa ja niiden käyttöönottoa harkitaan.

7 Loppupäätelmät

Tämän insinöörityön tarkoituksena oli tuottaa dokumentti, jossa käsitellään mobiilin monialustasovelluksen kehittämisen haasteita teknisen laadunvarmistuksen näkökulmasta. Työn pohjana käytettiin CGI Suomi Oy:n tuottamaa Mobilog-työnohjaussovellusta sekä erityisesti siihen kuuluvaa HTML5-mobiilisovellusta. Selvitystyön keskeisenä teemana oli etsiä ja kehittää testaukseen ja laadunvarmistukseen toimivat menetelmät, jotka pätevät myös monialustaisessa sovelluskehityksessä. Kokonaisuutena työ tarjoaa hyvän lähtökohdan esiteltyjen menetelmien jatkokehitykselle sekä niiden jalostamiseen sovelluskehityksen tueksi.

Suurimpana yksittäisenä haasteena tätä työtä tehdessä oli *HTML5*-sovelluksen toteuttamisen WP8-alustalle, sillä alustan rajapinnat ovat varsin suljetut. Lopulta rajoittunut tekniikka saneli ratkaisun ehdot ja joudumme nyt puntaroimaan koko työnohjaussovelluksen toteutuksen muutosta, jotta *NFC*-toiminnallisuus voidaan toteuttaa poiketen nykyisestä mallista. Haasteita aiheutti myös sovelluskäännöksen käyttötestit, sillä sovellusta pystyi ajamaan ainoastaan emulaattorissa johtuen lisenssisyistä. Sovelluksen varsinainen käyttökokemus jäi vielä vähäiseksi, joten testausta tullaan jatkamaan vielä tämän insinöörityön jälkeenkin. Vaikka sovelluskäännös ei ole vielä valmis, tulen jatkamaan kehitystyötä täysipainoisen sovellusratkaisun kehittämiseksi.

Esitellyt laadunvarmistusmenetelmät on suunniteltu suoraan CGI:n Mobilog-sovellusta varten, mutta ne on helposti toteutettavissa muissakin ohjelmistoprojekteissa. Työssä esiteltyjen menetelmien avulla voimme jatkossa kehittää Mobilog-laadunvarmistustyötä niin yhden kuin useammankin alustaratkaisun kohdalla. Seuraavana askeleena tämän työn jälkeen on esiteltyjen menetelmien jatkokehitys sekä niiden valjastaminen käytännön toteutukseen. Työ on kokonaisuudessaan onnistunut kokonaisuus ja täyttää sille määrittelemäni tavoitteet, vaikkakin *WP8*-sovelluksen käännöstyö on vielä kesken.

Lähteet

1 Cross-platform. en.wikipedia.org; 2014. Saatavilla osoitteesta: http://en.wikipedia.org/wiki/Cross-platform [viitattu 27.3.2014].

2 HTML5. fi.wikipedia.org; 2014. Saatavilla osoitteesta: http://fi.wikipedia.org/wiki/HTML5 [viitattu 19.3.2014].

3 Android apk. fi.wikipedia.org; 2014. Saatavilla osoitteesta: http://en.wikipedia.org/wiki/APK_(file_format) [viitattu 29.3.2014].

4 Android Development. fi.wikipedia.org; 2014. Saatavilla osoitteesta: http://en.wikipedia.org/wiki/Android_software_development [viitattu 1.4.2014].

5 Windows Phone 8. fi.wikipedia.org; 2014. Saatavilla osoitteesta: http://fi.wikipedia.org/wiki/Windows_Phone [viitattu 1.4.2014].

6 Apple. fi.wikipedia.org; 2014. Saatavilla osoitteesta: http://en.wikipedia.org/wiki/IOS [viitattu 1.4.2014].

7 BSD. fi.wikipedia.org; 2014. Saatavilla osoitteesta: http://fi.wikipedia.org/wiki/BSD-lisenssi [viitattu 1.4.2014].

8 Firefox OS. fi.wikipedia.org; 2014. Saatavilla osoitteesta: http://fi.wikipedia.org/wiki/Firefox_OS [viitattu 21.4.2014].

9 Jolla. https://jolla.com/fi/; 2014. Saatavilla osoitteesta: https://jolla.com/fi/ [viitattu 2.4.2014].

10 Natiivi vs HTML5. http://67.prosenttia.fi/2013/05/27/natiivi-hybridi-ja-html5/; 2014. Saatavilla osoitteesta: http://67.prosenttia.fi/2013/05/27/natiivi-hybridi-ja-html5/ [viitattu 7.4.2014].

11 Software testing. en.wikipedia.org; 2014. Saatavilla osoitteesta: http://en.wikipedia.org/wiki/Software_testing [viitattu 7.4.2014].

12 Erich Gamma RjJ Richard Helm. Design Patterns: Elements of Reusable Object-Oriented Software. Yhdysvallat; 1994.

13 SQA. en.wikipedia.org; 2014. Saatavilla osoitteesta: http://en.wikipedia.org/wiki/Software_quality_assurance [viitattu 23.3.2014].

14 Daniel Galin. Software Quality Assurance. Englanti; 2004.

15 Scrum. en.wikipedia.org; 2014. Saatavilla osoitteesta: http://en.wikipedia.org/wiki/Scrum_(software_development) [viitattu 7.4.2014].

16 Code Coverage. en.wikipedia.org; 2014. Saatavilla osoitteesta: http://en.wikipedia.org/wiki/Code_coverage [viitattu 8.4.2014].

17 Test-Driven Development. en.wikipedia.org; 2014. Saatavilla osoitteesta: http://en.wikipedia.org/wiki/Test-driven_development [viitattu 9.4.2014].

18 Extreme Programming. en.wikipedia.org; 2014. Saatavilla osoitteesta: http://en.wikipedia.org/wiki/Extreme_programming [viitattu 2.4.2014].

19 MSDN NDEF. msdn.microsoft.com; 2014. Saatavilla osoitteesta: http://msdn.microsoft.com/en-us/library/windows/apps/br241250.aspx?cs-save-lang=1&cs-lang=csharp#code-snippet-1 [viitattu 10.4.2014].

20 Nokia NFC. developer.nokia.com; 2014. Saatavilla osoitteesta: http://developer.nokia.com/community/wiki/Use_NFC_tags_with_Windows_Phone_8 [viitattu 10.4.2014].

Liitteet

Liite1: Haastattelun muistiinpanot

Aihe: *Testauksen kehittäminen ja laadun varmistus*
Haastateltava: *Toni Kapiainen (Testausasiantuntija)*
Päiväys:*14.3.2014*

Sprint -toiminnan kehittäminen?

- Pidemmät Sprintit ja versiojulkaisuja harvemmin.
- Sprinttien pidentäminen antaisi enemmän aikaa perusteelliselle testaukselle.
- Testaukseen käytettävää aikaa suhteessä kehitykseen olisi nostettava.

Miten testaus liittyy laadunvarmistamiseen sovelluksessa?

Testaus kulkee osana kehitystä ja hyväksymistestaus suoritetaan aina Sprinttien lopussa. Testauksella voidaan varmistaa tuotteen toimivuus ja sitä kautta tekninen laadukkuus. Testauksen pohjalta voidaan asettaa myös laatukriteereitä, joihin voidaan nojata määrittäessä laadudokumentaatiota.

Millä testausmenetelmillä pyrimme varmistamaan sovelluksen laadun?

Feature testaus:

- Kesto noin kolme viikkoa.
- Testataan vuorollaan kutakin ominaisuutta eristettynä muusta toiminnallisuudesta.
- Feature testaukseen liittyy Jenkins-automaatiotestit, sekä yksikkötestit.
- Mobiilisovelluksen yksikkötestit on toteutettu Jasmine-yksikkötestien avulla.

Yleisellä tasolla:

- Hyväksymistestauksessa testataan kaikki toiminnallisuudet yhdessä. Tarkoitus löytää ristiriitoja toiminnallisuuksien välillä.
- Kesto noin viikon.
- Käytetään myös apuna Jenkinsiä, sekä yksikkötestejä.

Mitä monialustatestaus vaatii testaukselta?

Monialustatestaus:

- Automaattittisen testauksen käyttöönotto suotavaa. Menetelmä mahdollistaa usean alustaratkaisun testaamisen yhdellä kertaa.
- Eri alustatoteutuksien arkkitehtuurit olisi hyvä olla samankaltaiset. Tämä tekee testauksesta helpompaa.
- Valmiiksi ohjelmoidut testitapaukset.
- Vaihtoehtoja työkaluiksi: SilkTest, HP Qtp, Selenium ja TestComplete.
- Yksikään toiminnallisuus ei ole hyväksytty, ennenkuin kaikki alustaratkaisut läpäisevät testit.
- Puhelimen UI-testausen käyttöönotto. Menetelmä tarjoaa mahdollisuuden hoitaa osan käyttötesteistä automaattisesti.
- Emulaattoreiden hyödyntäminen testausprosessissa. Menetelmä mahdollistaa testaamisen ilman fyysistä päätelaitetta.

Laadun mittarit, jotka sovelluksen on täytettävä?

- Testien kattavuus. Kuinka paljon koodin eri haaroja ajetaan läpi. Monialustaisuudessa äärimmäisen tärkeää, sillä testattavan koodin määrä kasvaa huomattavasti aiemmasta.
- Testauksen keskittäminen asiakkaan tarpeisiin. Testataan erityisesti niitä toiminnallisuuksia ja osa-alueita, joita asiakas käyttää eniten sovelluksessa.
- Nostetaan tuotteen perusominaisuudet ensisijaisen testauksen kohteeksi.
- Integraatioiden testaus. Integraatiot vaikuttavat suoraan puhelimen toimintaan ja tiedon välittämiseen.
- Simuloidaan testeissä järjestelmän käyttöä oikeassa työssä/kentällä.
- Virheindeksi. Kuinka paljon virheitä vanhassa versiossa on verraten uuteen. Mistä bugit ovat tulleet? Kuinka vakavia bugit ovat toiminnan kannalta? Tuotetaan dokumentti virheistä jokaisessa sprintissä.